Adobe Flex Tutorial

ADOBE FLEX TUTORIAL

Simply Easy Learning by tutorialspoint.com

tutorialspoint.com

ABOUT THE TUTORIAL

Adobe Flex Tutorial

Flex is a powerful, open source application framework that allows you to easily build mobile applications for iOS, Android, and BlackBerry Tablet OS devices, as well as traditional applications for browser and desktop using the same programming model, tool, and codebase.

You will build Flex applications more easily using Adobe Flash Builder software, an enterprise-class Eclipse based IDE.This tutorial will give you great understanding on Flex concepts needed to get a web and mobile application up and running.

Audience

This tutorial is designed for Software Professionals who are willing to learn Flex Programming in simple and easy steps. This tutorial will give you great understanding on Flex Programming concepts and after completing this tutorial you will be at intermediate level of expertise from where you can take yourself at higher level of expertise.

Prerequisites

Before proceeding with this tutorial you should have a basic understanding of any programming language, text editor and execution of programs etc. Because we are going to develop web based applications usingFlex, so it will be good if you have understanding on other web technologies like, HTML, CSS, AJAX etc.

Copyright & Disclaimer Notice

Table of Contents

CHAPTER

1

Flex-Overview

This chapter describes the basic definition and concepts of Adobe Flex::.

What is Flex?

FLEX is an open source application framework.

- Flex is a powerful, open source application framework which allows to build traditional applications for browser, mobile and desktop using the same programming model, tool, and codebase.
- Flex provides FLEX SDK consisting of the Flex class library (ActionScript classes), the Flex compilers, the debugger, the MXML and ActionScript programming languages, and other utilities to build expressive and interactive rich internet applications (RIA)
- Flex takes care of the user interface (UI) or the client-side functionality of a web application. Server-side functionality dependent on server-side components written in a traditional scripting language (Java/ PHP etc.)
- A Flex based application actually delivered as a SWF file and it closely resembles the HTML / Javascript portion of a traditional web application.
- Flex application is deployed as SWF file(s) plus an HTML wrapper, the CSS file(s) and any server-side script files (i.e. Java, .CFM, .PHP, etc) to the server. Like traditional web applications
- These resources are delivered from the server to the client browser using the customary HTTP request / response fashion and Flash Player run the application in the browser.

Advantages of Flex

- Flex applications being Flash Player based can access device capabilities like GPS, camera, local database, graphics accelerometer.
- Flex applications can run on Andriod, BlackBerry Tablet OS, iOS devices.

- Flex applications can run on Browsers as well as on Desktop.
- Flex applications are platform independent. UI can be native to platform or can be made same on each platform.
- Flex applications can interact with server with all major server side technologies like Java, Spring, Hibernate, PHP, Ruby, .NET, Adobe ColdFusion, and SAP using industry standards such as REST, SOAP, JSON, JMS, and AMF.
- Flex Applications developed assures Rich User Experience through intuitive interaction with the application and presenting information in a visually richer interface.
- Flex application is a single page application where states can can transition from one state to other state without having to fetch a new page from the server or to refresh the browser.
- Flex application reduces the load on the server to great extent because it is only required return the application once, rather than a new page every time when the user changes views.

Disadvantages of Flex

- Flex applications are single threaded applications but Flex provides an asynchronous programming model to mitigate this concern.
- Flex is actionscript and XML based. Learning of these two is a must to work in Flex.

CHAPTER

2

Flex - Environment Setup

This chapter describes the environment setup of Adobe Flex::.

This tutorial will guide you on how to prepare a development environment to start your work with Adobe Flex Framework. This tutorial will also teach you how to setup JDK and Adobe Flash Builder on your machine before you setup Flex Framework:

System Requirement

FLEX requires JDK 1.4 or higher so the very first requirement is to have JDK installed in your machine.

JDK	1.4 or above.
Memory	no minimum requirement.
Disk Space	no minimum requirement.
Operating System	no minimum requirement.

Follow the given steps to setup your environment to start with Flex application development.

Step 1 - Verify Java installation on your machine

Now open console and execute the following java command.

OS	**Task**	**Command**
Windows	Open Command Console	c:\> java -version
Linux	Open Command Terminal	$ java -version
Mac	Open Terminal	machine:~ joseph$ java -version

Let's verify the output for all the operating systems:

OS	**Generated Output**

Windows	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
Linux	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
Mac	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java HotSpot(TM)64-Bit Server VM (build 17.0-b17, mixed mode, sharing)

Step 2 - Setup Java Development Kit (JDK):

If you do not have Java installed then you can install the Java Software Development Kit (SDK) from Oracle's Java site: Java SE Downloads. You will find instructions for installing JDK in downloaded files, follow the given instructions to install and configure the setup. Finally set PATH and JAVA_HOME environment variables to refer to the directory that contains java and javac, typically java_install_dir/bin and java_install_dir respectively.

Set the **JAVA_HOME** environment variable to point to the base directory location where Java is installed on your machine. For example

OS	Output
Windows	Set the environment variable JAVA_HOME to C:\Program Files\Java\jdk1.6.0_21
Linux	export JAVA_HOME=/usr/local/java-current
Mac	export JAVA_HOME=/Library/Java/Home

Append Java compiler location to System Path.

OS	Output
Windows	Append the string ;%JAVA_HOME%\bin to the end of the system variable, Path.
Linux	export PATH=$PATH:$JAVA_HOME/bin/
Mac	not required

Step 3 - Setup Adobe Flash Builder 4.5

All the examples in this tutorial have been written using Adobe Flash Builder 4.5 Profession IDE Trial Version. So I would suggest you should have latest version of Adobe Flash Builder installed on your machine based on your operating system.

To install Adobe Flash Builder IDE, download the latest Adobe Flash Builder binaries fromhttp://www.adobe.com/in/products/flash-builder.html. Once you downloaded the installation, unpack the binary distribution into a convenient location. For example in C:\flash-builder on windows, or /usr/local/flash-builder on Linux/Unix and finally set PATH variable appropriately.

Flash Builder can be started by executing the following commands on windows machine, or you can simply double click on FlashBuilder.exe

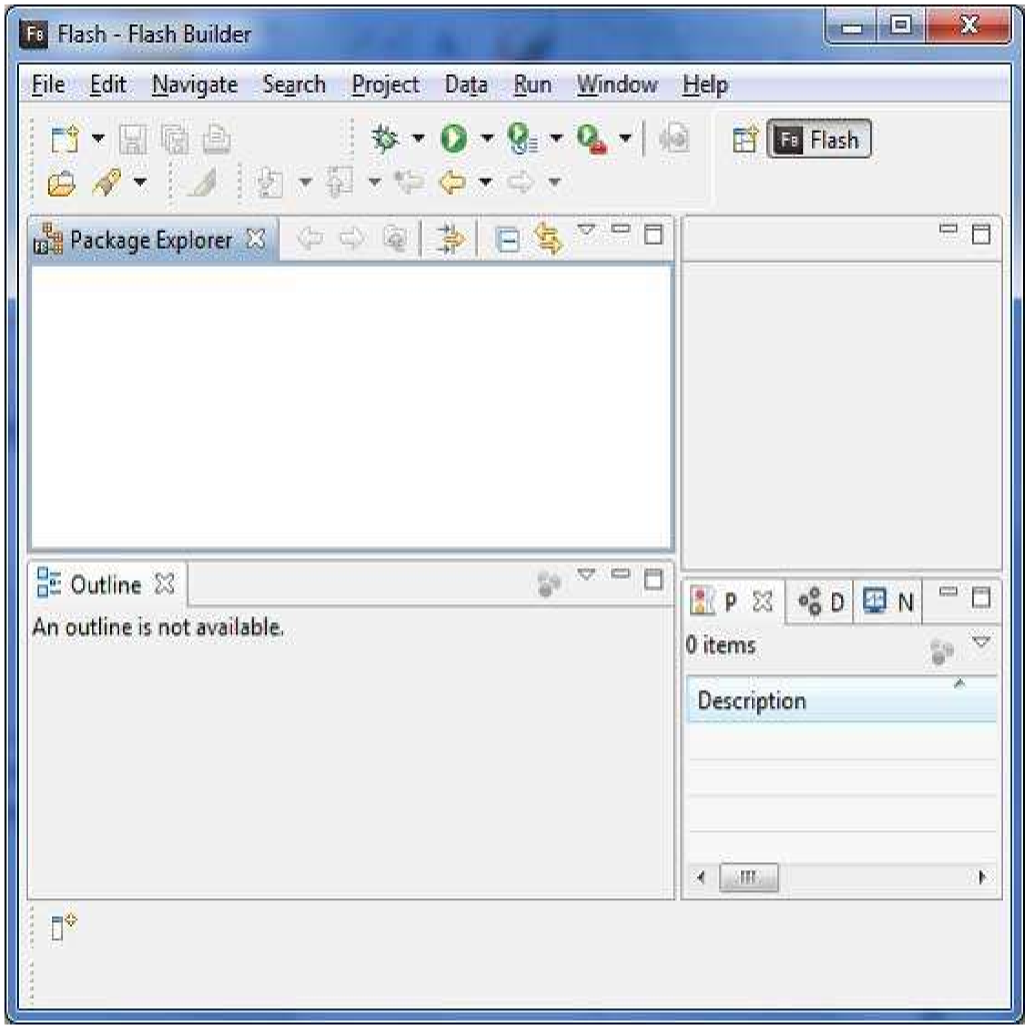
```
%C:\flash-builder\FlashBuilder.exe
```

Flash Builder can be started by executing the following commands on Unix (Solaris, Linux, etc.) machine:

```
$/usr/local/flash-builder/FlashBuilder
```

Adobe Flash Builder Trial Version can be used for 60 days. Just accept the terms and conditions and skip the initial registration steps and continue with the IDE. We're using the trial version for teaching purpose.

After a successful startup, if everything is fine then it should display following result:



An outline is not available.

Adobe Flash Builder comes pre-configured with FLEX SDKs. We're using FLEX SDK 4.5 in our examples which comes bundled with Adobe Flash Builder 4.5.

Step 4: Setup Apache Tomcat:

You can download the latest version of Tomcat from http://tomcat.apache.org/. Once you downloaded the installation, unpack the binary distribution into a convenient location. For example in **C:\apache-tomcat-6.0.33** on windows, or **/usr/local/apache-tomcat-6.0.33** on Linux/Unix and set CATALINA_HOME environment variable pointing to the installation locations.

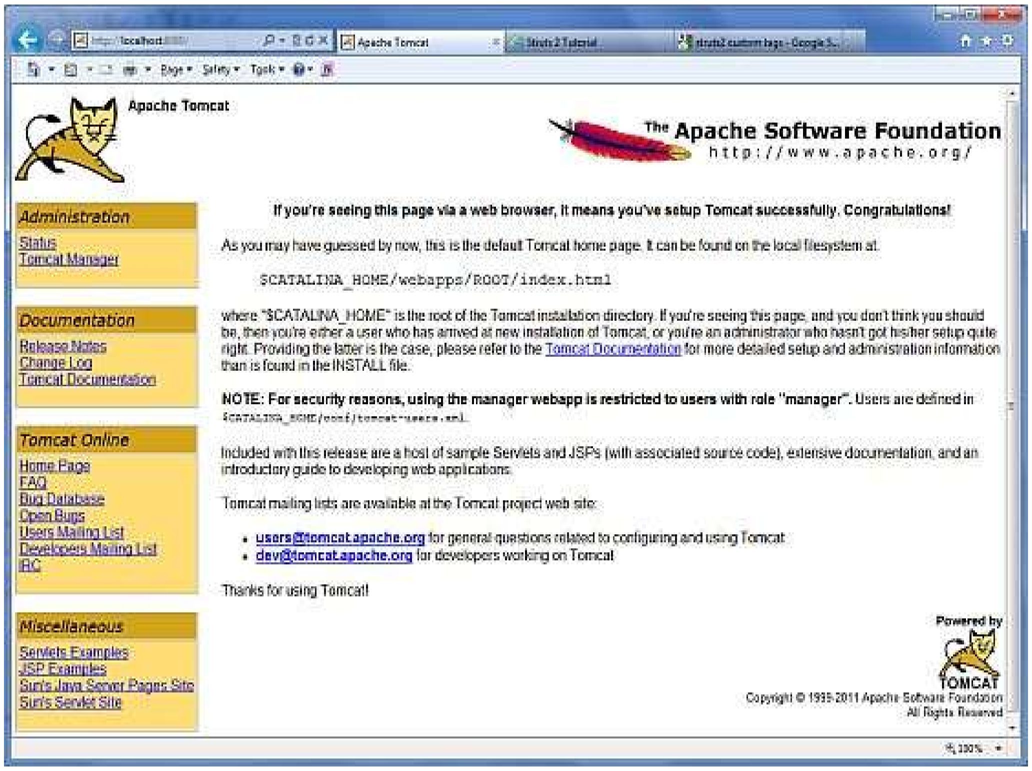
Tomcat can be started by executing the following commands on windows machine, or you can simply double click on startup.bat

```
%CATALINA_HOME%\bin\startup.bat

or

 C:\apache-tomcat-6.0.33\bin\startup.bat
```

Tomcat can be started by executing the following commands on Unix (Solaris, Linux, etc.) machine:

```
$CATALINA_HOME/bin/startup.sh

or

/usr/local/apache-tomcat-6.0.33/bin/startup.sh
```

After a successful startup, the default web applications included with Tomcat will be available by visiting http://localhost:8080/. If everything is fine then it should display following result:



If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

```
$CATALINA_HOME/webapps/ROOT/index.html
```

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the Tomcat Documentation for more detailed setup and administration information than is found in the INSTALL file.

NOTE: For security reasons, using the manager webapp is restricted to users with role "manager". Users are defined in `$CATALINA_HOME/conf/tomcat-users.xml`.

Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation, and an introductory guide to developing web applications.

Tomcat mailing lists are available at the Tomcat project web site:

- users@tomcat.apache.org for general questions related to configuring and using Tomcat
- dev@tomcat.apache.org for developers working on Tomcat

Thanks for using Tomcat!

Copyright © 1999-2011 Apache Software Foundation
All Rights Reserved

Further information about configuring and running Tomcat can be found in the documentation included here, as well as on the Tomcat web site: http://tomcat.apache.org

Tomcat can be stopped by executing the following commands on windows machine:

```
%CATALINA_HOME%\bin\shutdown

or

C:\apache-tomcat-5.5.29\bin\shutdown
```

Tomcat can be stopped by executing the following commands on Unix (Solaris, Linux, etc.) machine:

```
$CATALINA_HOME/bin/shutdown.sh
```

```
or

/usr/local/apache-tomcat-5.5.29/bin/shutdown.sh
```

CHAPTER

3

Flex - Applications

This chapter describes the applications under Adobe Flex:

Bfore we start with creating actual *HelloWorld* application using Flash Builder, let us see what are the actual parts of a Flex application. A Flex application consists of following four important parts out of which last part is optional but first three parts are mandatory:

- Flex Framework Libraries
- Client-side code
- Public Resources (HTML/JS/CSS)
- Server-side code

Sample locations of different parts of a typical Flex application HelloWord will be as shown below:

Name	**Location**
Project root	HelloWorld/
Flex Framework Libraries	Build Path
Public resources	html-template
Client-side code	src/com/tutorialspoint/client
Server-side code	src/com/tutorialspoint/server

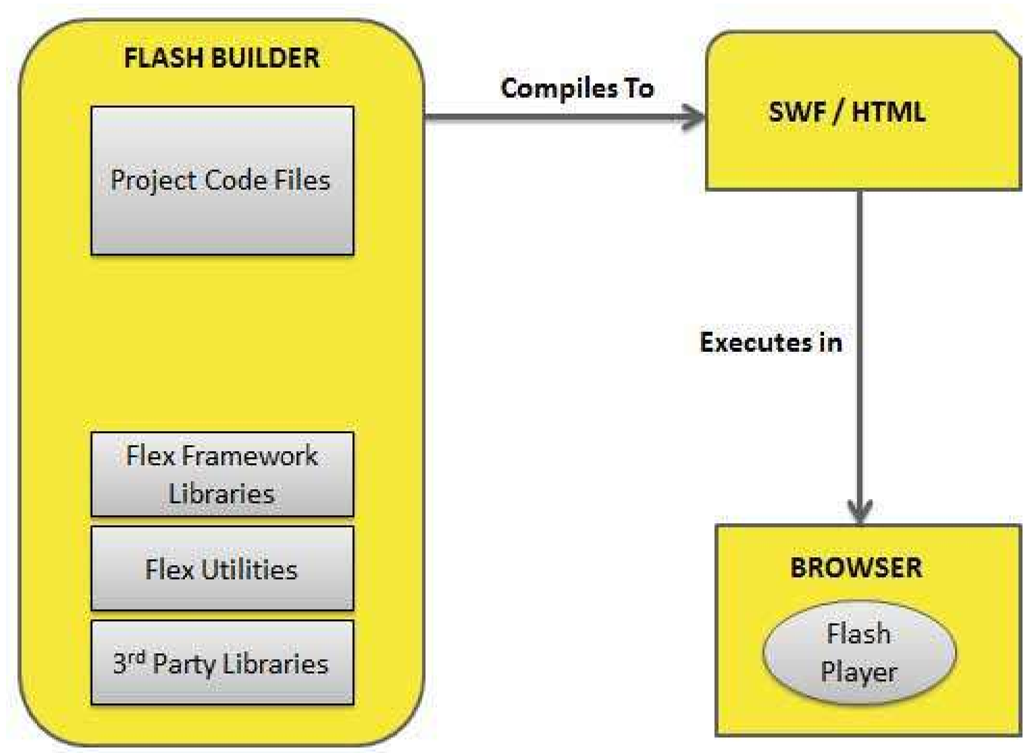
Application Build Process

Flex application required Flex Framework libraries. Flash Builder automatically add the libraries to build path.

When we build our code using Flash Builder, Flash builder will do the following tasks

- Compiles the source code to HelloWorld.swf file.

- Compiles a HelloWorld.html (a wrapper file for swf file) from a file index.template.html stored in html-template folder
- Copies HelloWorld.swf and HelloWorld.html files in target folder, bin-debug.
- Copies swfobject.js, a javascript code responsible to load swf file dynamically in HelloWorld.html in target folder, bin-debug
- Copies framework libraries in form of swf file named frameworks_xxx.swf in target folder, bin-debug
- Copies other flex modules (.swf files such as sparkskins_xxx.swf,textLayout_xxx.swf) in target folder.



Application Launch Process

- Open the HelloWorld.html file available in \HelloWorld\bin-debug folder in any web-browser.
- HelloWorld.swf will load automatically and application will start running.

Flex Framework Libraries

Following is the brief detail about few important framework libraries.

In flex libraries are denoted using .swc notation

S.N.	Nodes & Description
1	**playerglobal.swc** This library is specific to FlashPlayer installed on your machine and contains native methods supported by flash player.

2	**textlayout.swc** This library supports the text layout related features.
3	**framework.swc** This is the flex framework library contains the core features of Flex.
4	**mx.swc** This library stores the definations of mx UI controls.
5	**charts.swc** This library supports the charting controls.
6	**spark.swc** This library stores the definations of spark UI controls.
7	**sparkskins.swc** This library supports the skinning of spark UI controls.

Client-side code

Flex application code can be written in MXML and ActionScript.

S.N.	Type & Description
1	**MXML** MXML is an XML markup language that we'll use to lay out user interface components.MXML is compiled into ActionScript during build process.
2	**ActionScript** ActionScript is an object-oriented procedural programming language and is based on the ECMAScript (ECMA-262) edition 4 draft language specification.

In Flex, we can mix ActionScript and MXML, to do the following:

- Lay out user interface components using MXML tags
- Use MXML to declaratively define nonvisual aspects of an application, such as access to data sources on the server
- Use MXML to create data bindings between user interface components and data sources on the server.
- Use ActionScript to define event listeners inside MXML event attributes.
- Add script blocks using the <mx:Script> tag.
- Include external ActionScript files.
- Import ActionScript classes.
- Create ActionScript components.

Public resources

These are helper files referenced by Flex application, such as Host HTML page, CSS or images located under html-template folder.It contains following files

S.N.	File Name & Description
1	**index.template.html** Host HTML page, with place holders. Flash Builder uses this template to build actual page HelloWorld.html with HelloWorld.swf file.
2	**playerProductInstall.swf** This is a flash utility to install Flash Player in express mode.
3	**swfobject.js** This is the javascript responsible to check version of flash player installed and to load HelloWorld.swf in HelloWorld.html page.
4	**html-template/history** This folder contains resources for history management of the application.

HelloWorld.mxml

This is the actual MXML/AS (ActionScript) code written implementing the business logic of the application and that the Flex compiler translates into SWF file which will be executed by flash player in the browser.A sample HelloWorld Entry class will be as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Applicationxmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx"
width="100%"height="100%"
minWidth="500"minHeight="500"
initialize="application_initializeHandler(event)">

<fx:Script>
<![CDATA[
import mx.controls.Alert;
import mx.events.FlexEvent;
protected function btnClickMe_clickHandler(event:MouseEvent):void
         {
Alert.show("Hello World!");
         }

protected function  application_initializeHandler(event:FlexEvent):void
         {
            lblHeader.text = "My Hello World Application";
         }
      ]]>
</fx:Script>
<s:VGrouphorizontalAlign="center"width="100%"height="100%"
paddingTop="100"gap="50">
<s:Labelid="lblHeader"fontSize="40"color="0x777777"/>
<s:Buttonlabel="Click Me!"id="btnClickMe"
click="btnClickMe_clickHandler(event)"/>
</s:VGroup>
</s:Application>
```

Following Table gives the description of all the tags used in the above code script.

S.N.	Node & Description
1	**Application** Defines the Application container that is always the root tag of a Flex application.
2	**Script** Contains the business logic in ActionScript language.
3	**VGroup** Defines a Vertical Grouping Container which can contain Flex UI controls in vertical fashion.
4	**Label** Represents a Label control, a very simple user interface component that displays text.
5	**Button** Represents a Button control, which can be clicked to do some action.

Server-side code

This is the server side part of your application and its very much optional. If you are not doing any backend processing with-in your application then you do not need this part, but if there is some processing required at backend and your client-side application interact with the server then you will have to develop these components.

Next chapter will make use of all the above mentioned concepts to create HelloWorld application using Flash Builder.

CHAPTER

4

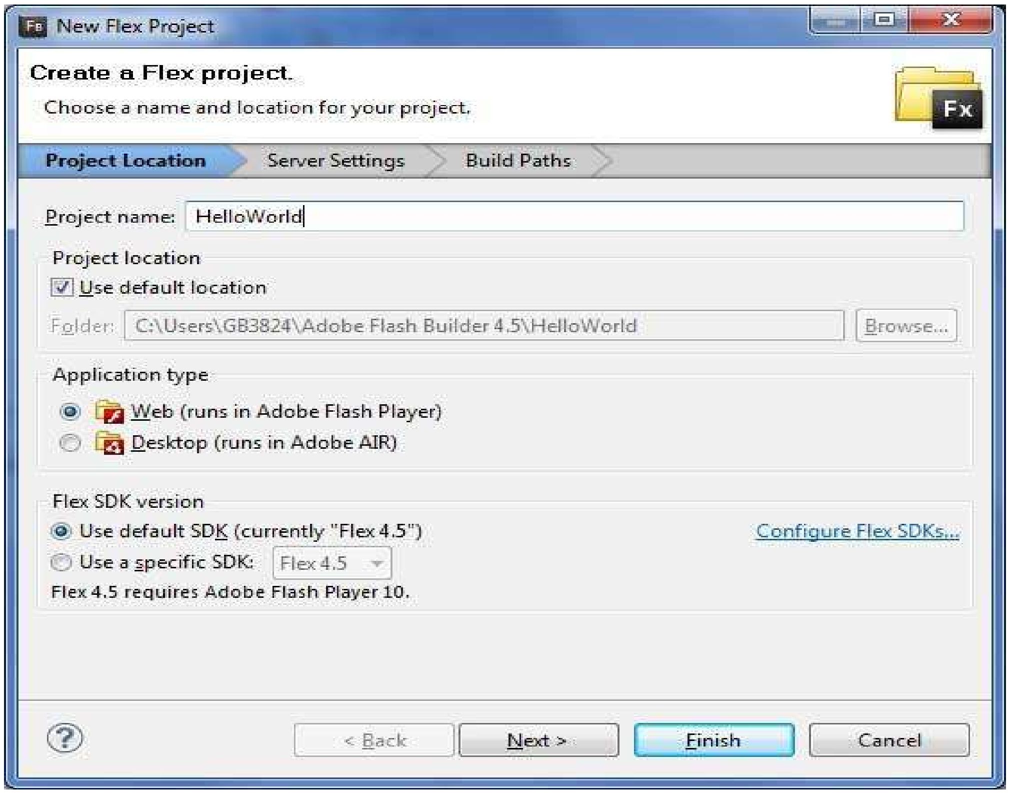
Flex – Create Application

This chapter shows how to create application under Adobe Flex.

We'll use Flash Builder 4.5 to create Flex Applications. Let's start with a simple *HelloWorld* application:

Step 1 - Create Project

The first step is to create a simple Flex Project using Flash Builder IDE. Launch project wizard using the option **File > New > Flex Project**. Now name your project as *HelloWorld* using the wizard window as follows:



Create a Flex project.

Choose a name and location for your project.

Project name: HelloWorld

Project location

☑ Use default location

Folder: C:\Users\GB3824\Adobe Flash Builder 4.5\HelloWorld

Application type

● Web (runs in Adobe Flash Player)

○ Desktop (runs in Adobe AIR)

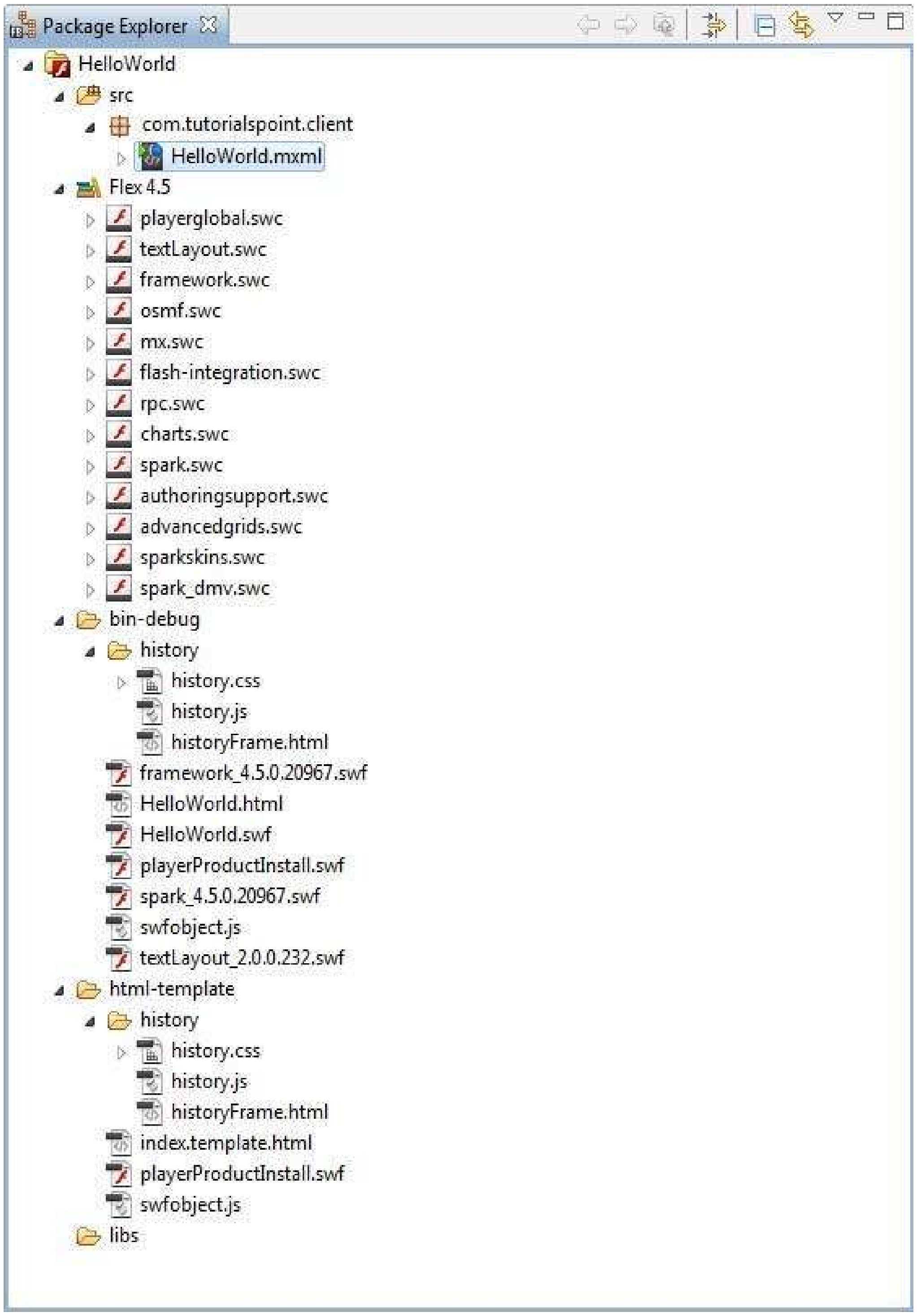
Flex SDK version

● Use default SDK (currently "Flex 4.5")

○ Use a specific SDK: Flex 4.5

Flex 4.5 requires Adobe Flash Player 10.

Select Application Type **Web (runs in Adobe Flash Player)** if not selected and leave other default values as such and click Finish Button. Once your project is created successfully, you will have following content in your Project Explorer:



```
HelloWorld
  src
    com.tutorialspoint.client
      HelloWorld.mxml
  Flex 4.5
    playerglobal.swc
    textLayout.swc
    framework.swc
    osmf.swc
    mx.swc
    flash-integration.swc
    rpc.swc
    charts.swc
    spark.swc
    authoringsupport.swc
    advancedgrids.swc
    sparkskins.swc
    spark_dmv.swc
  bin-debug
    history
      history.css
      history.js
      historyFrame.html
    framework_4.5.0.20967.swf
    HelloWorld.html
    HelloWorld.swf
    playerProductInstall.swf
    spark_4.5.0.20967.swf
    swfobject.js
    textLayout_2.0.0.232.swf
  html-template
    history
      history.css
      history.js
      historyFrame.html
    index.template.html
    playerProductInstall.swf
    swfobject.js
  libs
```

Here is brief description of all important folders:

Folder	Location
src	Source code (mxml / as classes) files. We've created com/tutorialspoint/client folder structure containing the client-side specific java classes responsible for client UI display.
bin-debug	This is the output part, it represents the actual deployable web application. history folder contains support files for history management of Flex application. framework_xxx.swf, flex framework files to be used by flex application. HelloWorld.html, wrapper/host HTML File for flex application. HelloWorld.swf, our flex based application. playerProductInstall.swf, flash player express installer. spark_xxx.swf, library for spark component support. swfobject.js, javascript responsible to load HelloWorld.swf in HelloWorld.html. It checks flash player version and passes initialization parameter to HelloWorld.swf file. textLayout_xxx.swf, library for text component support.
html-template	This represents the configurable web application. Flash Builder compiles files from html-template to bin-debug folder. history folder contains support files for history management of Flex application. index.template.html, wrapper/host HTML File for flex application having place holders for Flash Builder specific configuration. Gets compiled to HelloWorld.html in bin-debug folder during build. playerProductInstall.swf, flash player express installer.Gets copied to bin-debug folder during build. swfobject.js, javascript responsible to load HelloWorld.swf in HelloWorld.html. It checks flash player version and passes initialization parameter to HelloWorld.swf file. Gets copied to bin-debug folder during build.

Step 2 - Create external CSS file

Create a CSS file styles.css for Wrapper HTML page in html-template folder.

```
html, body  {
   height:100%;
}
body {
   margin:0;
   padding:0;
   overflow:auto;
   text-align:center;
}
object:focus {
   outline:none;
}
#flashContent {
   display:none;
}

.pluginHeader {
   font-family:Arial, Helvetica, sans-serif;
   font-size:14px;
   color:#9b1204;
   text-decoration:none;
   font-weight:bold;
}

.pluginInstallText {
   font-family:Arial, Helvetica, sans-serif;
   font-size:12px;
   color:#000000;
   line-height:18px;
   font-style:normal;
}

.pluginText {
   font-family:Arial, Helvetica, sans-serif;
   font-size:12px;
   color:#000000;
   line-height:18px;
   font-style:normal;
}
```

Step 3 - Modify Wrapper HTML page template

Modify Wrapper HTML page template **index.template.html** in **html-template** folder. Flash Builder will create a default Wrapper HTML page template *html-template/index.template.html*, which will be compiled to HelloWorld.html. This file contains placeholders which Flash Builder replaces during compilation process for example flash player version, application name etc.

Let us modify this file to display custom messages if flash plugin is not installed.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
```

```
<head>
<title>${title}</title>
<meta name="google" value="notranslate" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<link rel="stylesheet" href="styles.css" type="text/css"></link>
<link rel="stylesheet" type="text/css" href="history/history.css" />
<script type="text/javascript" src="history/history.js"></script>
<script type="text/javascript" src="swfobject.js"></script>
<script type="text/javascript">
   // For version detection, set to min. required Flash Player version,
   //or 0 (or 0.0.0), for no version detection.
   var swfVersionStr = "${version_major}.${version_minor}.${version_revision}";
   // To use express install, set to playerProductInstall.swf,
   //otherwise the empty string.
   var xiSwfUrlStr = "${expressInstallSwf}";
   var flashvars = {};
   var params = {};
   params.quality = "high";
   params.bgcolor = "${bgcolor}";
   params.allowscriptaccess = "sameDomain";
   params.allowfullscreen = "true";
   var attributes = {};
   attributes.id = "${application}";
   attributes.name = "${application}";
   attributes.align = "middle";
   swfobject.embedSWF(
   "${swf}.swf", "flashContent",
   "${width}", "${height}",
   swfVersionStr, xiSwfUrlStr,
   flashvars, params, attributes);
   // JavaScript enabled so display the flashContent div in case
   //it is not replaced with a swf object.
   swfobject.createCSS("#flashContent", "display:block;text-align:left;");
</script>
</head>
<body>
<div id="flashContent">
   <p style="margin:100px;">
   <table  width="700" cellpadding="10" cellspacing="2" border="0">
      <tr><td class="pluginHeader">Flash Player Required</td></tr>
      <tr><td class="pluginText">The Adobe Flash Player version
      10.2.0 or greater is required.</td></tr>
      <tr><td class = "pluginInstallText" align="left">
         <table border="0" width="100%">
         <tr class = "pluginInstallText" >
         <td>Click here to download and install Adobe Flash Player:</td>
         <td> </td>
         <td align="right">     <script type="text/javascript">
         var pageHost
         =((document.location.protocol == "https:") ? "https://" : "http://");
         document.write("<a target='_blank'"
         +" href='http://get.adobe.com/flashplayer/'><"
         +"img style='border-style: none' src='"
         +pageHost +"www.adobe.com/images/shared/download_buttons/get_flash_player.gif'"
         +" alt='Get Adobe Flash player' /></a>" );
         </script>
         </td>
         </tr>
         </table>
         </td>
      </tr>
   </table>
   </p>
</div>
```

```
<noscript>
   <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
   width="${width}" height="${height}" id="${application}">
   <param name="movie" value="${swf}.swf" />
   <param name="quality" value="high" />
   <param name="bgcolor" value="${bgcolor}" />
   <param name="allowScriptAccess" value="sameDomain" />
   <param name="allowFullScreen" value="true" />
   <!--[if !IE]>-->
   <object type="application/x-shockwave-flash" data="${swf}.swf"
   width="${width}" height="${height}">
   <param name="quality" value="high" />
   <param name="bgcolor" value="${bgcolor}" />
   <param name="allowScriptAccess" value="sameDomain" />
   <param name="allowFullScreen" value="true" />
   <!--<![endif]-->
   <!--[if gte IE 6]>-->
   <p>
      <p style="margin:100px;">
      <table  width="700" cellpadding="10" cellspacing="2" border="0">
      <tr><td class="pluginHeader">Flash Player Required</td></tr>
      <tr><td class="pluginText">The Adobe Flash Player version
      10.2.0 or greater is required.</td></tr>
      <tr><td class = "pluginInstallText" align="left">
         <table border="0" width="100%">
         <tr class = "pluginInstallText" >
         <td>Click here to download and install Adobe Flash Player:</td>
         <td> </td>
         <td align="right">
         <script type="text/javascript">
         var pageHost
         = ((document.location.protocol == "https:") ? "https://" : "http://");
         document.write("<a target='_blank'"
         +" href='http://get.adobe.com/flashplayer/'><"
         +"img style='border-style: none' src='"
         +pageHost
         +"www.adobe.com/images/shared/download_buttons/get_flash_player.gif'"
         +" alt='Get Adobe Flash player' /></a>" );
         </script>
         </td>
         </tr>
         </table>
         </td>
      </tr>
   </table>
   </p>
   </p>
   <!--<![endif]-->
     <p style="margin:100px;">
      <table  width="700" cellpadding="10" cellspacing="2" border="0">
      <tr><td class="pluginHeader">Flash Player Required</td></tr>
      <tr><td class="pluginText">The Adobe Flash Player version
      10.2.0 or greater is required.</td></tr>
      <tr><td class = "pluginInstallText" align="left">
         <table border="0" width="100%">
         <tr class = "pluginInstallText" >
         <td>Click here to download and install Adobe Flash Player:</td>
         <td> </td>
         <td align="right">     <script type="text/javascript">
         var pageHost
         = ((document.location.protocol == "https:") ? "https://" : "http://");
         document.write("<a target='_blank'"
         +" href='http://get.adobe.com/flashplayer/'><"
         +"img style='border-style: none' src='"
```

```
          +pageHost
          +"www.adobe.com/images/shared/download_buttons/get_flash_player.gif'"
          +" alt='Get Adobe Flash player' /></a>" );
          </script>
          </td>
          </tr>
          </table>
          </td>
       </tr>
    </table>
    </p>
    <!--[if !IE]>-->
    </object>
    <!--<![endif]-->
    </object>
</noscript>
</body>
</html>
```

Step 4 - Create internal CSS file

Create a CSS file **Style.css** for **HelloWorld.mxml** in **src/com/tutorialspoint** folder. Flex provides similar css styles for its UI Controls as there are css styles for HTML UI controls.

```
/* CSS file */
@namespace s "library://ns.adobe.com/flex/spark";
@namespace mx "library://ns.adobe.com/flex/mx";

.heading
{
   fontFamily: Arial, Helvetica, sans-serif;
   fontSize: 17px;
   color: #9b1204;
   textDecoration:none;
   fontWeight:normal;
}

.button {
   fontWeight: bold;
}

.container {
   cornerRadius :10;
   horizontalCenter :0;
   borderColor: #777777;
   verticalCenter:0;
   backgroundColor: #efefef;
}
```

Step 5 - Modify Entry Level Class

Flash Builder will create a default mxml file *src/com.tutorialspoint/HelloWorld.mxml*, which is having root tag <application> container for the application. Let us modify this file to display "Hello,World!":

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   width="100%" height="100%"
   minWidth="500" minHeight="500"
   initialize="application_initializeHandler(event)">
   <fx:Style source="/com/tutorialspoint/client/Style.css"/>
```

```
    <fx:Script>
       <![CDATA[
        import mx.controls.Alert;
        import mx.events.FlexEvent;
        protected function btnClickMe_clickHandler(event:MouseEvent):void
        {
           Alert.show("Hello World!");
        }

        protected function application_initializeHandler(event:FlexEvent):void
        {
           lblHeader.text = "My Hello World Application";
        }
       ]]>
    </fx:Script>
    <s:BorderContainer width="500" height="500" id="mainContainer"
       styleName="container">
       <s:VGroup width="100%" height="100%" gap="50" horizontalAlign="center"
          verticalAlign="middle">
          <s:Label id="lblHeader" fontSize="40" color="0x777777"
             styleName="heading"/>
          <s:Button label="Click Me!" id="btnClickMe"
             click="btnClickMe_clickHandler(event)" styleName="button" />
       </s:VGroup>
    </s:BorderContainer>
 </s:Application>
```

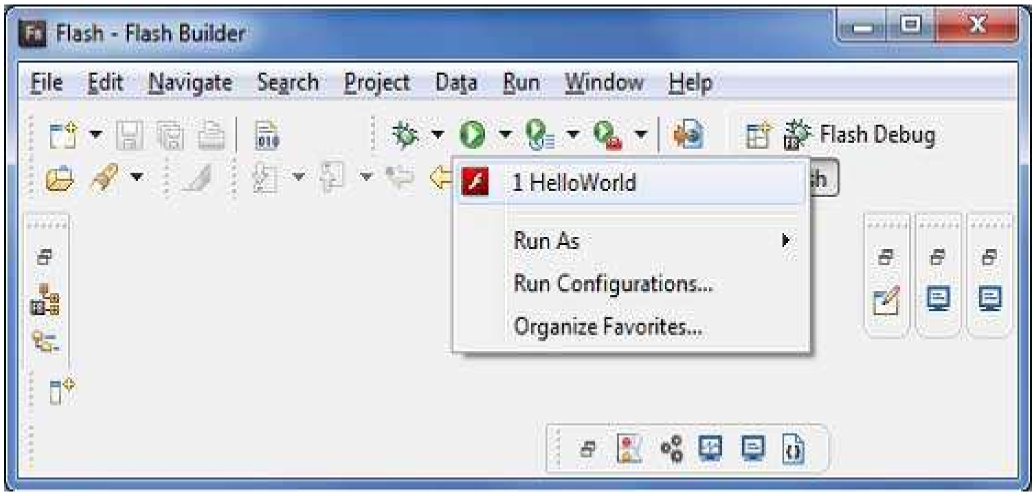
You can create more mxml or actionscript files in the same source directory to define either new applications or to define helper routines.

Step 6 - Build Application

Flash Builder has **Build Automatically** by default checked. Just check the **Problems** View if there is any error. Once you are done with the changes, you will not see any errors.

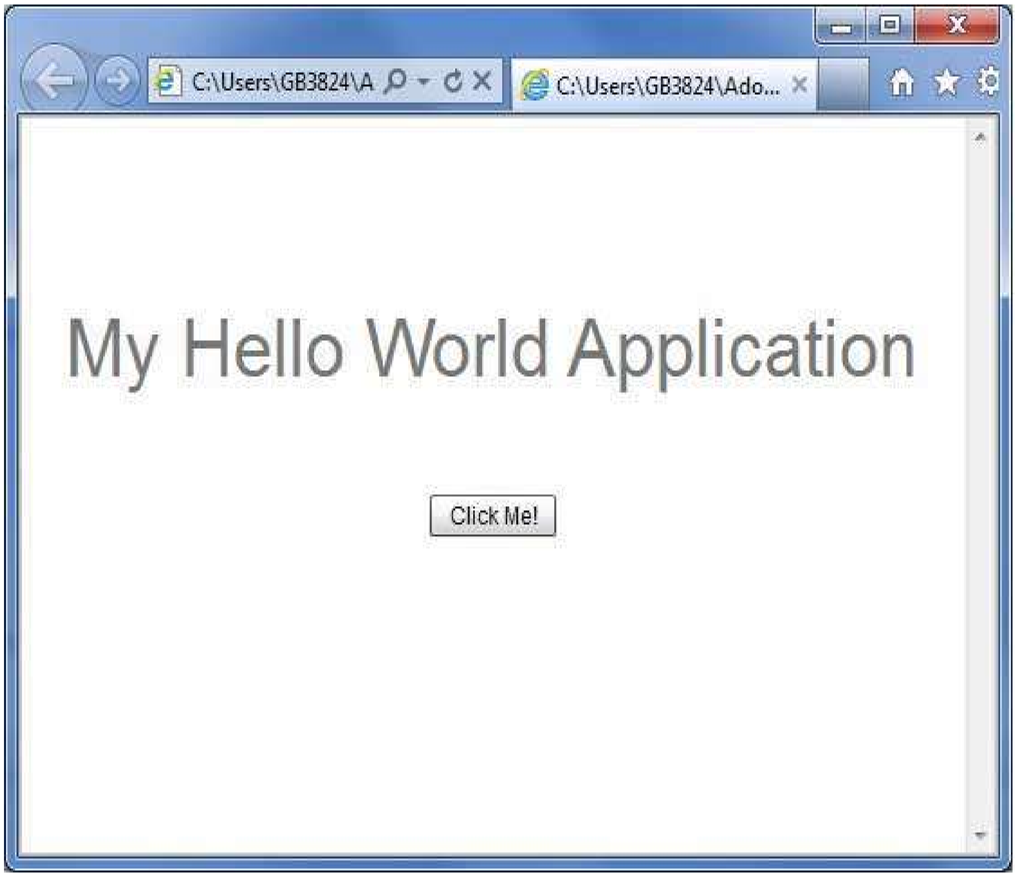
Step 7 - Run Application

Now click on Run application menu and select **HelloWorld** application to run the application.



If everything is fine, you must see browser pop up and application up and running. If everything is fine with your application, this will produce following result: [Try it online]

Because you are running your application in flash player, so you will need to install Flash Player plugin for your browser. Simply follow the onscreen instructions to install the plugin. If you already have Flash Player plugin set for your browser, then you should be able to see the following output:



Congratulations! you have implemented your first application using Flex.

CHAPTER

5

Flex – Deploy Application

This chapter shows the deployment of application:

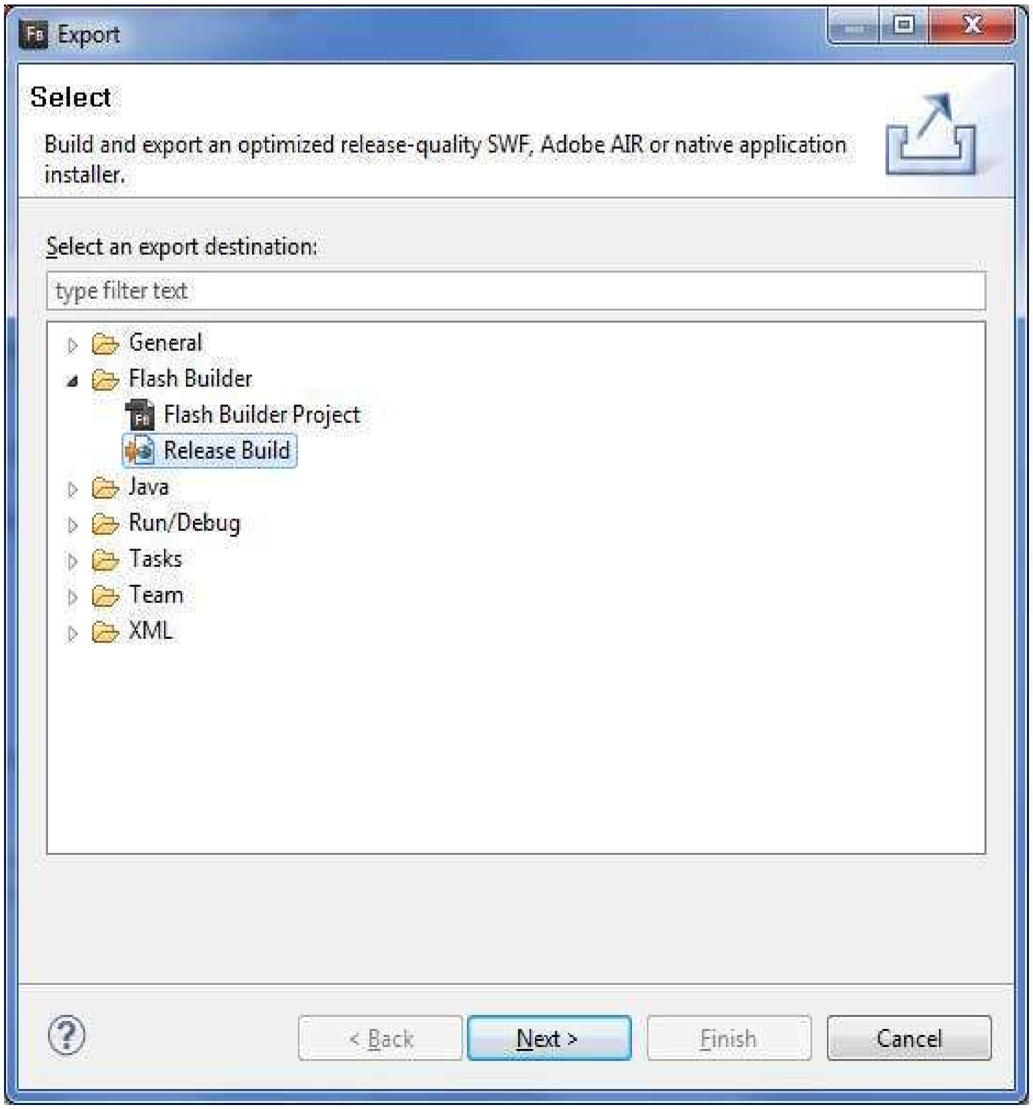
This tutorial will explain you how to create an application **war** file and how to deploy that in Apache Tomcat Websever root. If you understood this simple example then you will also be able to deploy a complex Flex application following the same steps.

Let us follow the following steps to create a Flex application:

Step	Description
1	Create a project with a name *HelloWorld* under a package *com.tutorialspoint.client* as explained in the *Flex - Create Application* chapter.
2	Modify *HelloWorld.mxml* as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

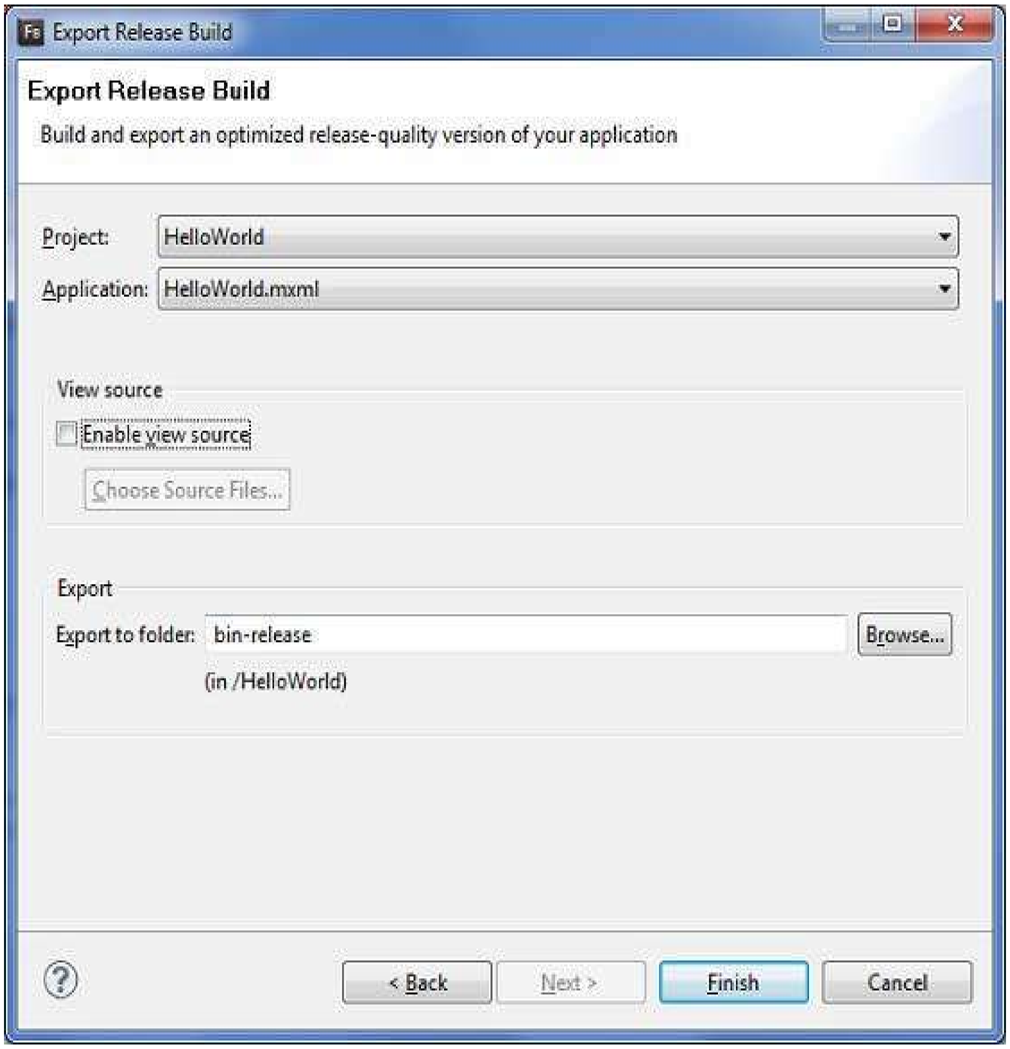
Follow the following steps to create a release build of a Flex application and then deploy it to tomcat server:

The first step is to create a release build using Flash Builder IDE. Launch release build wizard using the option **File > Export > Flash Builder > Release Build**



.

Select project as *HelloWorld* using the wizard window as follows



Leave other default values as such and click Finish Button. Now Flash Builder will create a bin-release folder containing the project's release build.

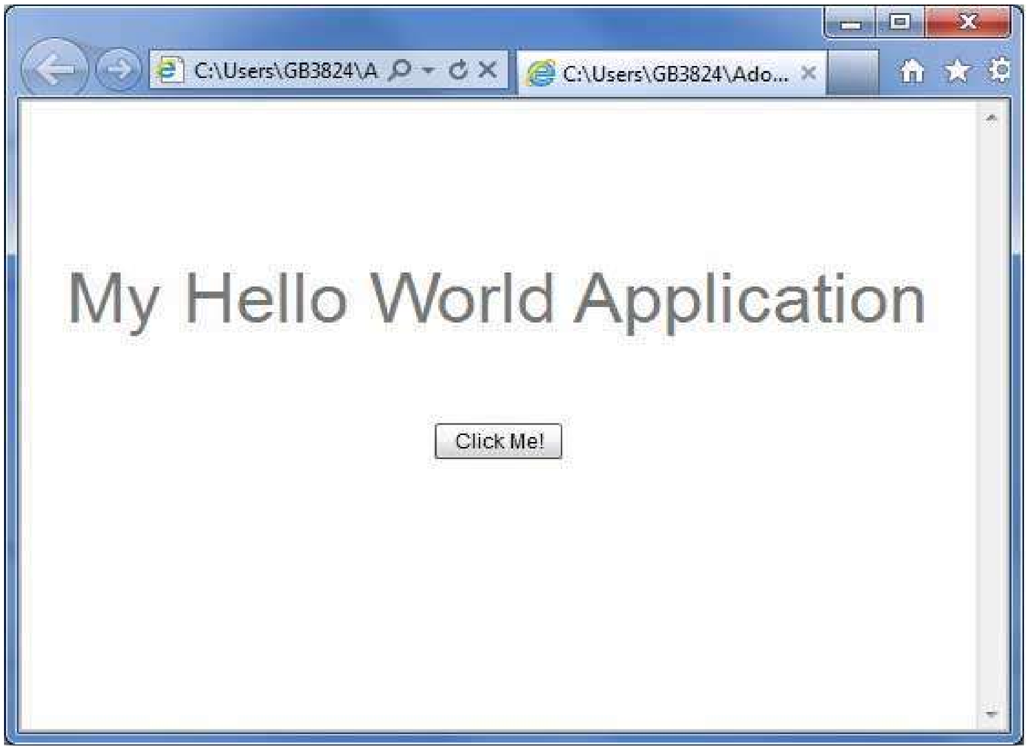
Now our release build is ready ,let us follow the following steps to deploy a Flex application:

Step	Description
1	Zip the content of the **bin-release** folder of the application in the form of HelloWorld.war file and deploy it in Apache Tomcat Webserver.
2	Launch your web application using appropriate URL as explained below in the last step.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   width="100%" height="100%"
   minWidth="500" minHeight="500"
   initialize="application_initializeHandler(event)">
   <fx:Style source="/com/tutorialspoint/client/Style.css"/>
   <fx:Script>
      <![CDATA[
         import mx.controls.Alert;
         import mx.events.FlexEvent;
         protected function btnClickMe_clickHandler(event:MouseEvent):void
         {
            Alert.show("Hello World!");
         }

         protected function application_initializeHandler(event:FlexEvent):void
         {
            lblHeader.text = "My Hello World Application";
         }
      ]]>
   </fx:Script>
   <s:BorderContainer width="500" height="500" id="mainContainer"
      styleName="container">
      <s:VGroup width="100%" height="100%" gap="50" horizontalAlign="center"
         verticalAlign="middle">
         <s:Label id="lblHeader" fontSize="40" color="0x777777"
            styleName="heading"/>
         <s:Button label="Click Me!" id="btnClickMe"
            click="btnClickMe_clickHandler(event)" styleName="button" />
      </s:VGroup>
   </s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, this will produce following result: [Try it online]



Create WAR File

Now our applictaion is working fine and we are ready to export it as a war file. Follow the following steps:

- Go into your project's **bin-release** directory **C:\workspace\HelloWorld\bin-release**
- Select all the files & folders available inside bin-release directory.
- Zip all the selected files & folders in a file called *HelloWorld.zip*.
- Rename *HelloWorld.zip* to *HelloWorld.war*.

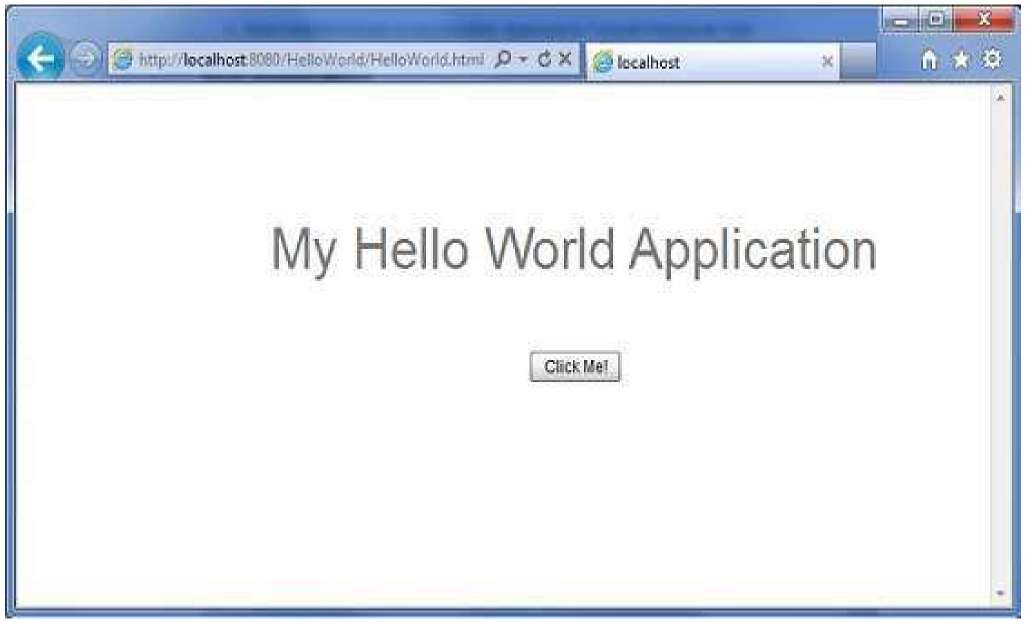
Deploy WAR file

- Stop the tomcat server.
- Copy the *HelloWorld.war* file to **tomcat installation directory > webapps folder.**
- Start the tomcat server.
- Look inside webapps directory, there should be a folder **HelloWorld** got created.
- Now HelloWorld.war is successfully deployed in Tomcat Webserver root.

Run Application

Enter a url in web browser: **http://localhost:8080/HelloWorld/HelloWorld.html** to launch the application

Server name (localhost) and port (8080) may vary as per your tomcat configuration.


http://localhost:8080/HelloWorld/HelloWorld.html
localhost
My Hello World Application
Click Me!

CHAPTER

6

Flex – Life Cycle Phases

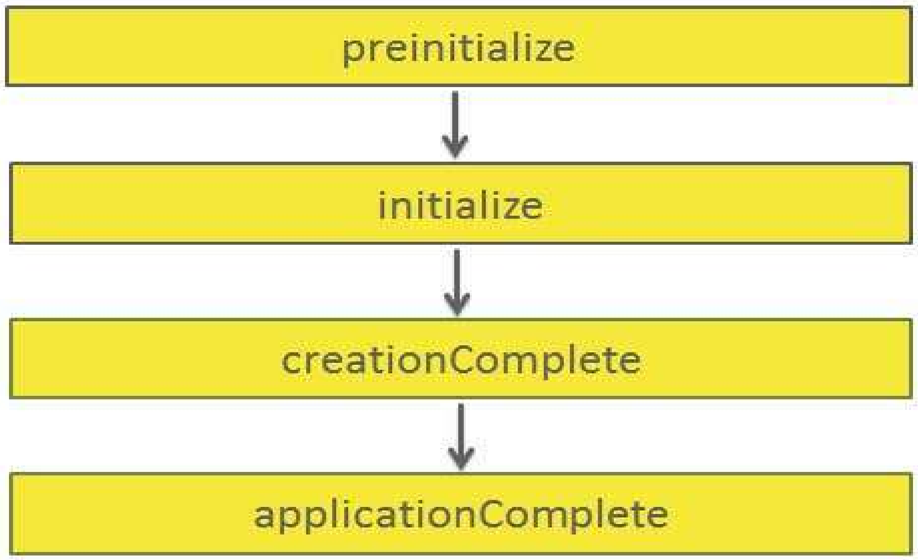
This chapter shows the phases under Life Cycle of Flex application:.

Life Cycle of flex application:

Although you can build Flex applications without understanding life cycle phases of an application life cycle, but it is good to know the basic mechanism: the order in which things occur.It will help you configure features such as load other Flex applications at runtime, and manage the process of loading and unloading class libraries and assets at runtime.

A good understanding of the Flex application life cycle will enable you to build better applications and optimize them because you will know where to optimally run code. For example, if you need to ensure that some code runs during a preloader, you need to know where to place the code for that event.

When we loads flex application in browser the following events occurs during the lifeCycle of flex application.



Following is the brief detail about different Flex Life Cycle Events.

S.N.	Event & Description

1	**preInitialize:** mx.core.UIComponent.preinitialize **Event Type:** mx.events.FlexEvent.PREINITIALIZE This event is dispatched at the beginning of the component initialization sequence. The component is in a very raw state when this event is dispatched. Many components, such as Button control, creates internal child components to implement functionality. For example, the Button control creates an internal UITextField component to represent its label text. When Flex dispatches the preinitialize event, the children, including all the internal children, of a component have not yet been created.
2	**initialize:** mx.core.UIComponent.initialize **Event Type:** mx.events.FlexEvent.INITIALIZE This event is dispatched after preinitialize phase. Flex framework initializes the internal structure of this component during this phase. This event automatically fires when the component is added to a parent. you do not need to call initialize() generally.
3	**creationComplete:** mx.core.UIComponent.creationComplete **Event Type:** mx.events.FlexEvent.CREATION_COMPLETE This event is dispatched when the component has finished its construction, property processing, measuring, layout, and drawing. At this point, depending on its visible property, the component is not visible even though it has been drawn.
4	**applicationComplete:** spark.components.Application.applicationComplete **Event Type:**mx.events.FlexEvent.APPLICATION_COMPLETE Dispatched after the Application has been initialized, processed by the LayoutManager, and attached to the display list. This is the last event of the application creation life cycle and signifies that application has been loaded completely.

Flex Life Cycle Example

Let us follow the following steps to test life cycle of a Flex application by creating a test application:

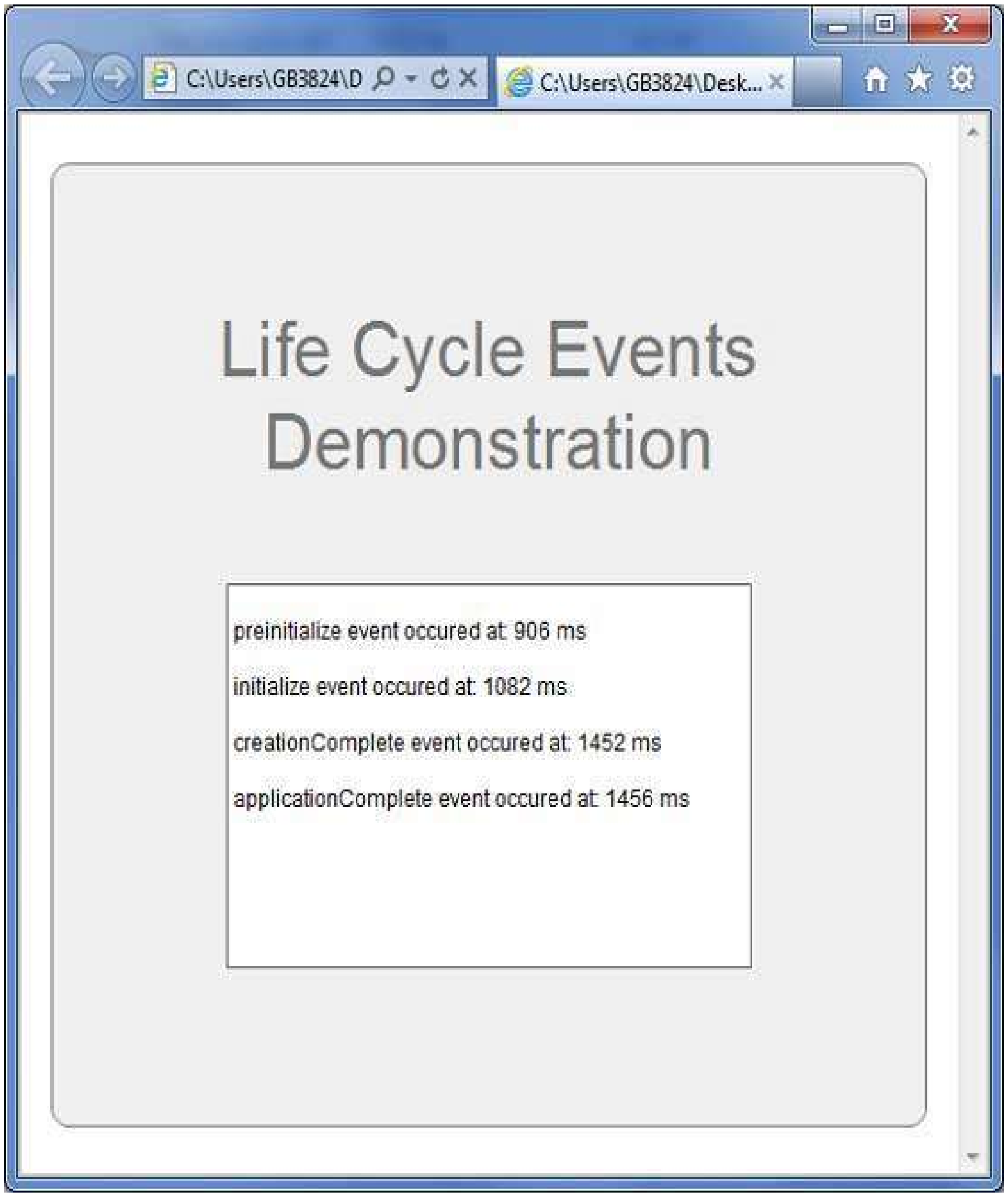
Step	Description

1	Create a project with a name *HelloWorld* under a package *com.tutorialspoint.client* as explained in the *Flex - Create Application* chapter.
2	Modify *HelloWorld.mxml* as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   width="100%" height="100%" minWidth="500" minHeight="500"
   initialize="reportEvent(event)"
   preinitialize="reportEvent(event)"
   creationComplete="reportEvent(event)"
   applicationComplete="reportEvent(event)">
   <fx:Style source="/com/tutorialspoint/client/Style.css"/>
   <fx:Script>
      <![CDATA[
         import mx.controls.Alert;
         import mx.events.FlexEvent;

         [Bindable]
         private var report:String = "";

         private function reportEvent(event:FlexEvent):void{
            report += "\n" + (event.type + " event occured at: "
            + getTimer() + " ms" + "\n");
         }
      ]]>
   </fx:Script>
   <s:BorderContainer width="500" height="500" id="mainContainer"
      styleName="container">
      <s:VGroup width="100%" height="100%" gap="50"
         horizontalAlign="center" verticalAlign="middle">
         <s:Label textAlign="center" width="100%" id="lblHeader"
         fontSize="40" color="0x777777" styleName="heading"
         text="Life Cycle Events Demonstration"/>
         <s:TextArea id="reportText" text="{report}" editable="false"
         width="300" height="200">
         </s:TextArea>
      </s:VGroup>
   </s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, this will produce following result: [Try it online]



Life Cycle Events Demonstration

```
preinitialize event occured at: 906 ms
initialize event occured at: 1082 ms
creationComplete event occured at: 1452 ms
applicationComplete event occured at: 1456 ms
```

CHAPTER

7

Flex – Style With CSS

This chapter describes the styles under Adobe Flex:

Flex supports the use of CSS syntax and styles to apply styles to its UI controls in the same way as CSS to HTML components.

Way #1: Using external style sheet file

You can refer to a style sheet available in the class path of the application. For example consider Style.css file in **com/tutorialspoint/client folder** where HelloWorld.mxml file also lies.

```
/* CSS file */
@namespace s "library://ns.adobe.com/flex/spark";
@namespace mx "library://ns.adobe.com/flex/mx";
...
.container {
   cornerRadius :10;
   horizontalCenter :0;
   borderColor: #777777;
   verticalCenter:0;
   backgroundColor: #efefef;
}
```

Then css file can be referred by following code snippet

```
<fx:Style source="/com/tutorialspoint/client/Style.css"/>
```

Assign styles to UI component using styleName property

```
<s:BorderContainer width="500" height="500" id="mainContainer"
      styleName="container">
...
</s:BorderContainer>
```

Way #2: Using styles within UI container component

You can define styles within UI container component using <fx:Style> tag

Class Level Selector

```
<fx:Style>
@namespace s "library://ns.adobe.com/flex/spark";
@namespace mx "library://ns.adobe.com/flex/mx";

/* class level selector  */
.errorLabel {
   color: red;
}
</fx:Style>
```

Assign styles to UI component using styleName property.

```
<s:Label id="errorMsg" text="This is an error message" styleName="errorLabel"/>
```

Id Level Selector

Style UI component using id selector.

```
<fx:Style>
/* id level selector  */
#msgLabel {
   color: gray;
}
</fx:Style>
<s:Label id="msgLabel" text="This is a normal message" />
```

Type Level Selector

Style one type of UI Component in one go.

```
<fx:Style>
/* style applied on all buttons  */
s|Button {
   fontSize: 15;
   color: #9933FF;
}
</fx:Style>
<s:Button label="Click Me!" id="btnClickMe"
click="btnClickMe_clickHandler(event)"  />
```

Flex Style with CSS Example

Let us follow the following steps to check css styling of a Flex application by creating a test application:

Step	Description

1	Create a project with a name *HelloWorld* under a package *com.tutorialspoint.client* as explained in the *Flex - Create Application* chapter.
2	Modify *Style.css*, *HelloWorld.mxml* as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

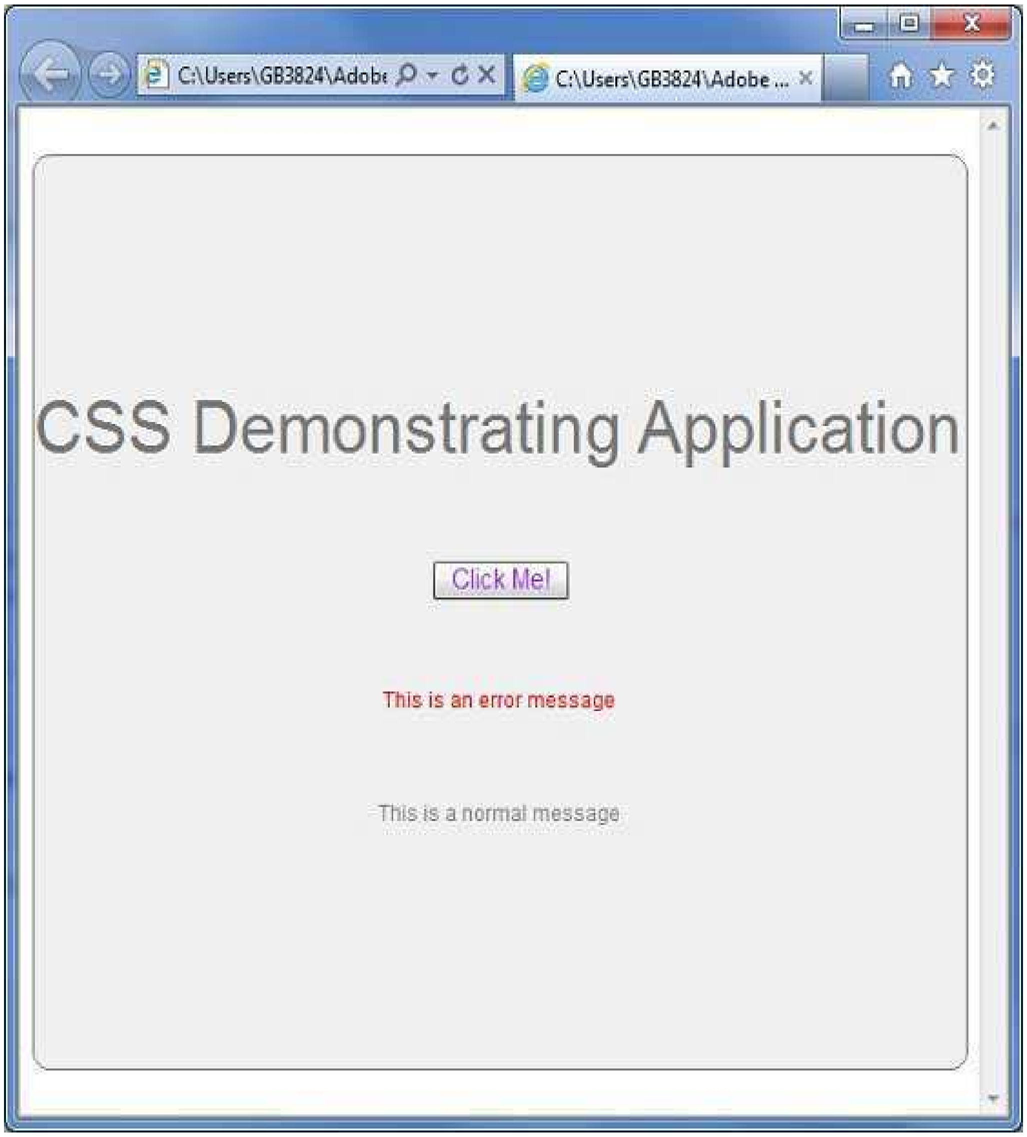
Following is the content of the modified css file **src/com.tutorialspoint/Style.css**.

```
/* CSS file */
@namespace s "library://ns.adobe.com/flex/spark";
@namespace mx "library://ns.adobe.com/flex/mx";

.heading
{
   fontFamily: Arial, Helvetica, sans-serif;
   fontSize: 17px;
   color: #9b1204;
   textDecoration:none;
   fontWeight:normal;
}

.button {
   fontWeight: bold;
}

.container {
   cornerRadius :10;
   horizontalCenter :0;
   borderColor: #777777;
   verticalCenter:0;
   backgroundColor: #efefef;
}
```

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   width="100%" height="100%" minWidth="500" minHeight="500"
   initialize="application_initializeHandler(event)">
   <!--Add reference to style sheet -->
   <fx:Style source="/com/tutorialspoint/client/Style.css"/>

   <!--Using styles within mxml file -->
   <fx:Style>
      @namespace s "library://ns.adobe.com/flex/spark";
      @namespace mx "library://ns.adobe.com/flex/mx";

      /* class level selector  */
      .errorLabel {
         color: red;
      }
      /* id level selector  */
      #msgLabel {
         color: gray;
      }

      /* style applied on all buttons  */
      s|Button {
         fontSize: 15;
         color: #9933FF;
      }
   </fx:Style>
   <fx:Script>
      <![CDATA[
         import mx.controls.Alert;
         import mx.events.FlexEvent;
         protected function btnClickMe_clickHandler(event:MouseEvent)
         :void {
            Alert.show("Hello World!");
         }

         protected function application_initializeHandler(event:FlexEvent)
         :void {
            lblHeader.text = "CSS Demonstrating Application";
         }
      ]]>
   </fx:Script>
   <s:BorderContainer width="560" height="500" id="mainContainer"
      styleName="container">
      <s:VGroup width="100%" height="100%" gap="50"
         horizontalAlign="center" verticalAlign="middle">
         <s:Label width="100%" id="lblHeader" fontSize="40"
            color="0x777777" styleName="heading"/>
         <s:Button label="Click Me!" id="btnClickMe"
            click="btnClickMe_clickHandler(event)"  />
         <s:Label id="errorMsg"
            text="This is an error message" styleName="errorLabel" />
         <s:Label id="msgLabel" text="This is a normal message" />
      </s:VGroup>
   </s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, this will produce following result: [Try it online]



CSS Demonstrating Application

This is an error message

This is a normal message

CHAPTER

8

Flex – Style With Skin

This chapter style with skinning in Adobe Flex:

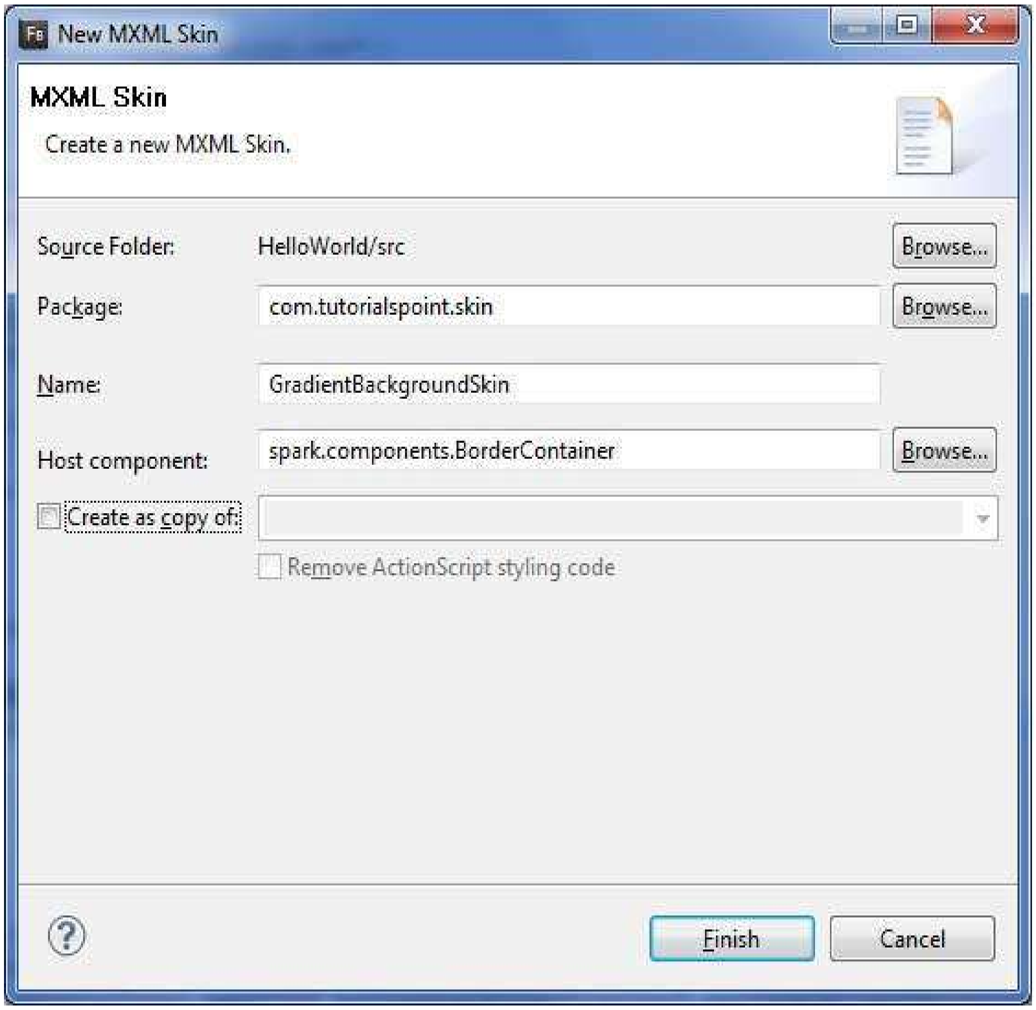
What is Skining?

Skinning in Flex, is a process of customizing look and feel of a UI Component completely.

- A Skin can define text, image, filters , transitions and states of a compoent.
- A Skin can be created as a seperate mxml or ActionScript component.
- Using skin, we can control all visual aspects of a UI component.
- The process of defining skin is same for all the UI component.

Step 1: Create a skin

Launch Create MXML Skin wizard using the option **File > New > MXML Skin**.



Source Folder: HelloWorld/src

Package: com.tutorialspoint.skin

Name: GradientBackgroundSkin

Host component: spark.components.BorderContainer

☐ Create as copy of:

☐ Remove ActionScript styling code

Enter Package as **com.tutorialspoint.skin**, name as **GradientBackgroundSkin** and choose host component as existing flex BorderContainer control **spark.component.BorderContainer**.

Now you've created a skin for a BorderContainer. Modify content of the mxml skin file**src/com.tutorialspoint/skin/GradientBackgroundSkin.mxml**. Update fill layer as follows:

```
<!-- fill -->
<s:Rect id="backgroundRect" left="0" right="0" height="100%" top="0">
   <s:fill>
      <s:LinearGradient rotation="90">
         <s:GradientEntry color="0x888888" ratio="0.2"/>
         <s:GradientEntry color="0x111111" ratio="1"/>
      </s:LinearGradient>
   </s:fill>
</s:Rect>
```

Step 2: Apply skin

You can apply skin over a component using two ways

Apply skin in MXML script (statically)

Apply **GradientBackgroundSkin** to a BorderContainer with id **mainContainer** using its **skinClass**attribute.

```
<s:BorderContainer width="560" height="500" id="mainContainer"
   styleName="container">
   <s:VGroup width="100%" height="100%" gap="50"
   horizontalAlign="center" verticalAlign="middle"
   skinClass="com.tutorialspoint.skin.GradientBackgroundSkin">
```

Apply skin in ActionScript (dynamically)

Apply **GradientBackgroundSkin** to a BorderContainer with id **mainContainer** using its **skinClass**property.

```
protected function gradientBackground_clickHandler(event:MouseEvent):void
{
   mainContainer.setStyle("skinClass", GradientBackgroundSkin );
}
```

Flex Style with Skin Example

Let us follow the following steps to see skinning in action in a Flex application by creating a test application:

Step	Description
1	Create a project with a name *HelloWorld* under a package *com.tutorialspoint.client* as explained in the *Flex - Create Application* chapter.
2	Create skin *GradientBackgroundSkin.mxml* under a package *com.tutorialspoint.skin* as explained above. Keep rest of the files unchanged.
3	Modify *HelloWorld.mxml* as explained below. Keep rest of the files unchanged.
4	Compile and run the application to make sure business logic is working as per the requirements.

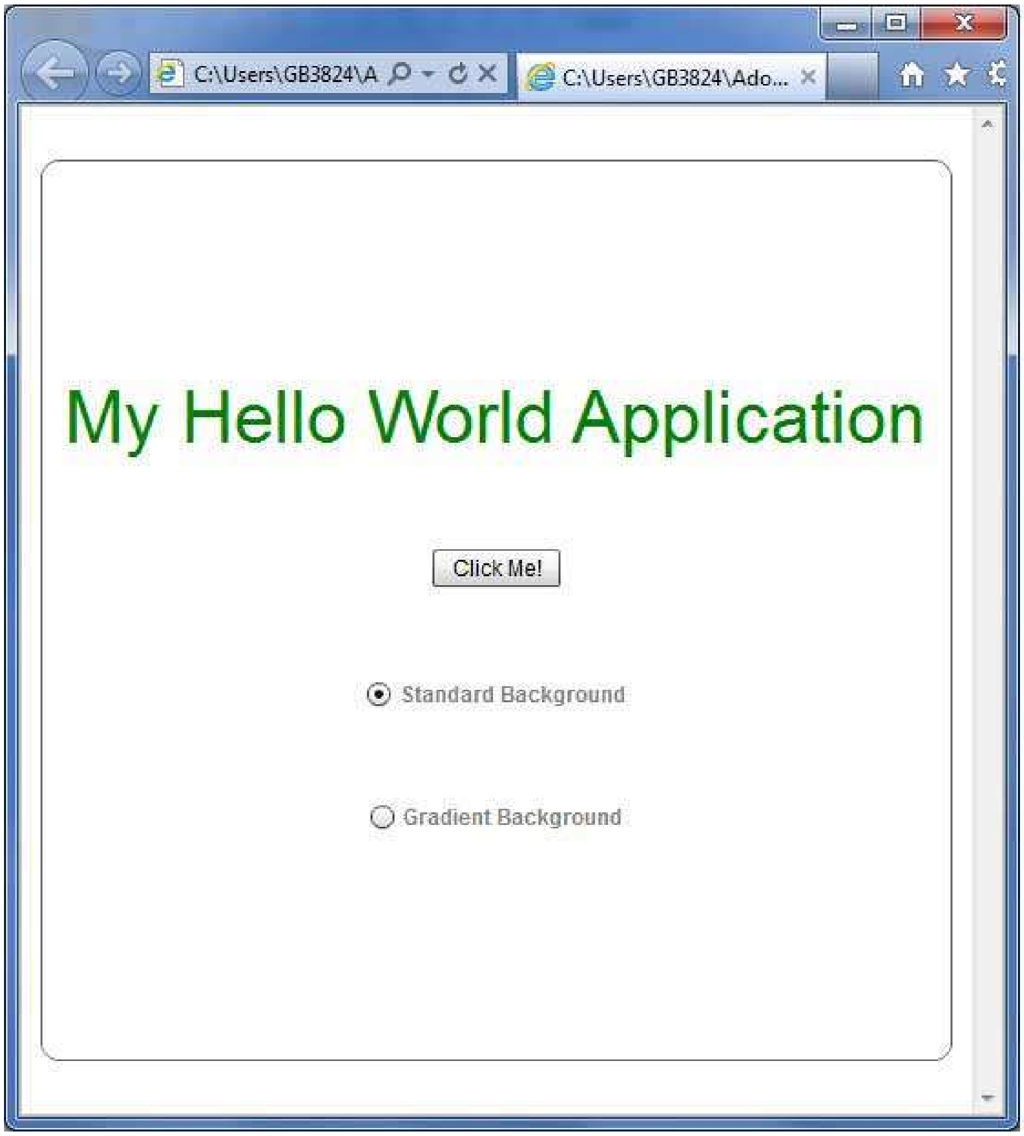
Following is the content of the **GradientBackgroundSkin.mxml** file**src/com/tutorialspoint/skin/GradientBackgroundSkin.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Skin xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx">
   <!-- host component -->
   <fx:Metadata>
      [HostComponent("spark.components.BorderContainer")]
   </fx:Metadata>

   <!-- states -->
   <s:states>
      <s:State name="disabled" />
      <s:State name="disabled" />
      <s:State name="normal" />
   </s:states>

   <!-- SkinParts
   name=contentGroup, type=spark.components.Group, required=false
   -->
   <!-- fill -->
   <s:Rect id="backgroundRect" left="0" right="0" height="100%" top="0">
      <s:fill>
         <s:LinearGradient rotation="90">
            <s:GradientEntry color="0x111111" ratio="0.2"/>
            <s:GradientEntry color="0x888888" ratio="1"/>
         </s:LinearGradient>
      </s:fill>
   </s:Rect>
   <!-- must specify this for the host component -->
   <s:Group id="contentGroup" left="0" right="0" top="0" bottom="0" />
</s:Skin>
```

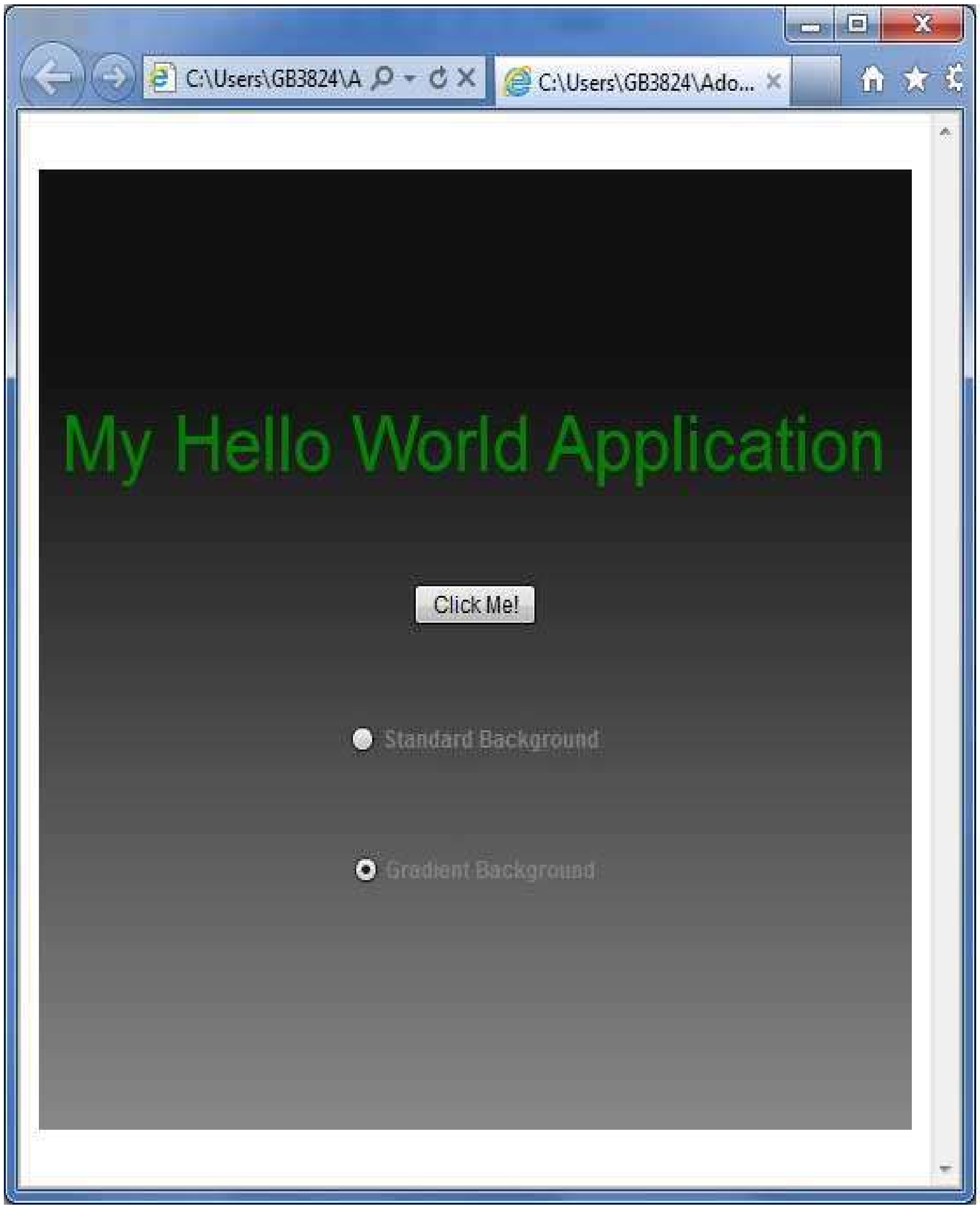
Following is the content of the modified **HelloWorld.mxml** file**src/com/tutorialspoint/client/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   width="100%" height="100%" minWidth="500" minHeight="500"
   initialize="application_initializeHandler(event)">

  <fx:Style source="/com/tutorialspoint/client/Style.css"/>

  <fx:Script>
  <![CDATA[
     import com.tutorialspoint.skin.GradientBackgroundSkin;
     import mx.controls.Alert;
     import mx.events.FlexEvent;
     import spark.skins.spark.BorderContainerSkin;

     protected function btnClickMe_clickHandler(event:MouseEvent):void
     {
        Alert.show("Hello World!");
     }

     protected function application_initializeHandler(event:FlexEvent):void
     {
        lblHeader.text = "My Hello World Application";
     }

     protected function gradientBackground_clickHandler(event:MouseEvent):void
     {
         mainContainer.setStyle("skinClass", GradientBackgroundSkin );
     }

     protected function standardBackground_clickHandler(event:MouseEvent):void
     {
        mainContainer.setStyle("skinClass", BorderContainerSkin );
     }
   ]]>
   </fx:Script>
   <fx:Declarations>
      <s:RadioButtonGroup id="selectorGroup" />
   </fx:Declarations>
   <s:BorderContainer width="500" height="500" id="mainContainer"
      skinClass="spark.skins.spark.BorderContainerSkin"
      horizontalCenter="0" verticalCenter="0" cornerRadius="10">
      <s:VGroup width="100%" height="100%" gap="50" horizontalAlign="center"
      verticalAlign="middle">
         <s:Label id="lblHeader" fontSize="40" color="green"
         styleName="heading"/>
         <s:Button label="Click Me!" id="btnClickMe"
            click="btnClickMe_clickHandler(event)"/>
         <s:RadioButton color="gray" fontWeight="bold"
            group="{selectorGroup}" label="Standard Background"
            click="standardBackground_clickHandler(event)" selected="true"/>
         <s:RadioButton color="gray" fontWeight="bold"
            group="{selectorGroup}" label="Gradient Background"
            click="gradientBackground_clickHandler(event)"/>
      </s:VGroup>
   </s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, this will produce following result: [Try it online]



My Hello World Application


C:\Users\GB3824\A
C:\Users\GB3824\Ado...
My Hello World Application
Click Me!
Standard Background
Gradient Background

CHAPTER

9

Flex – Data Binding

This chapter describes the Data Binding concepts.

What is Data Binding?

Data Binding is a process in which data of one object is tied to another object. Data binding requires a source property, a destination property and a triggering event which indicates when to copy the data from source to destination.

Flex provides three ways to do Data Binding

- Curly brace syntax in MXML Script ({})
- <fx:binding> tag in MXML
- BindingUtils in ActionScript

Data Binding - Using Curly Braces in MXML

Following example demonstrates using curly braces to specify data binding of a source to destination.

```
<s:TextInput id="txtInput1"/>
<s:TextInput id="txtInput2" text = "{txtInput1.text}"/>
```

Data Binding - Using <fx:Binding> tag in MXML

Following example demonstrates using <fx:Binding> tag to specify data binding of a source to destination.

```
<fx:Binding source="txtInput1.text" destination="txtInput2.text" />
<s:TextInput id="txtInput1"/>
<s:TextInput id="txtInput2"/>
```

Data Binding - Using BindingUtils in ActionScript

Following example demonstrates using BindingUtils to specify data binding of a source to destination.

```
<fx:Script>
   <![CDATA[
      import mx.binding.utils.BindingUtils;
      import mx.events.FlexEvent;

      protected function txtInput2_preinitializeHandler(event:FlexEvent):void
      {
         BindingUtils.bindProperty(txtInput2,"text",txtInput1, "text");
      }
]]>
</fx:Script>
<s:TextInput id="txtInput1"/>
<s:TextInput id="txtInput2"
preinitialize="txtInput2_preinitializeHandler(event)"/>
```

Flex Data Binding Example

Let us follow the following steps to see skinning in action in a Flex application by creating a test application:

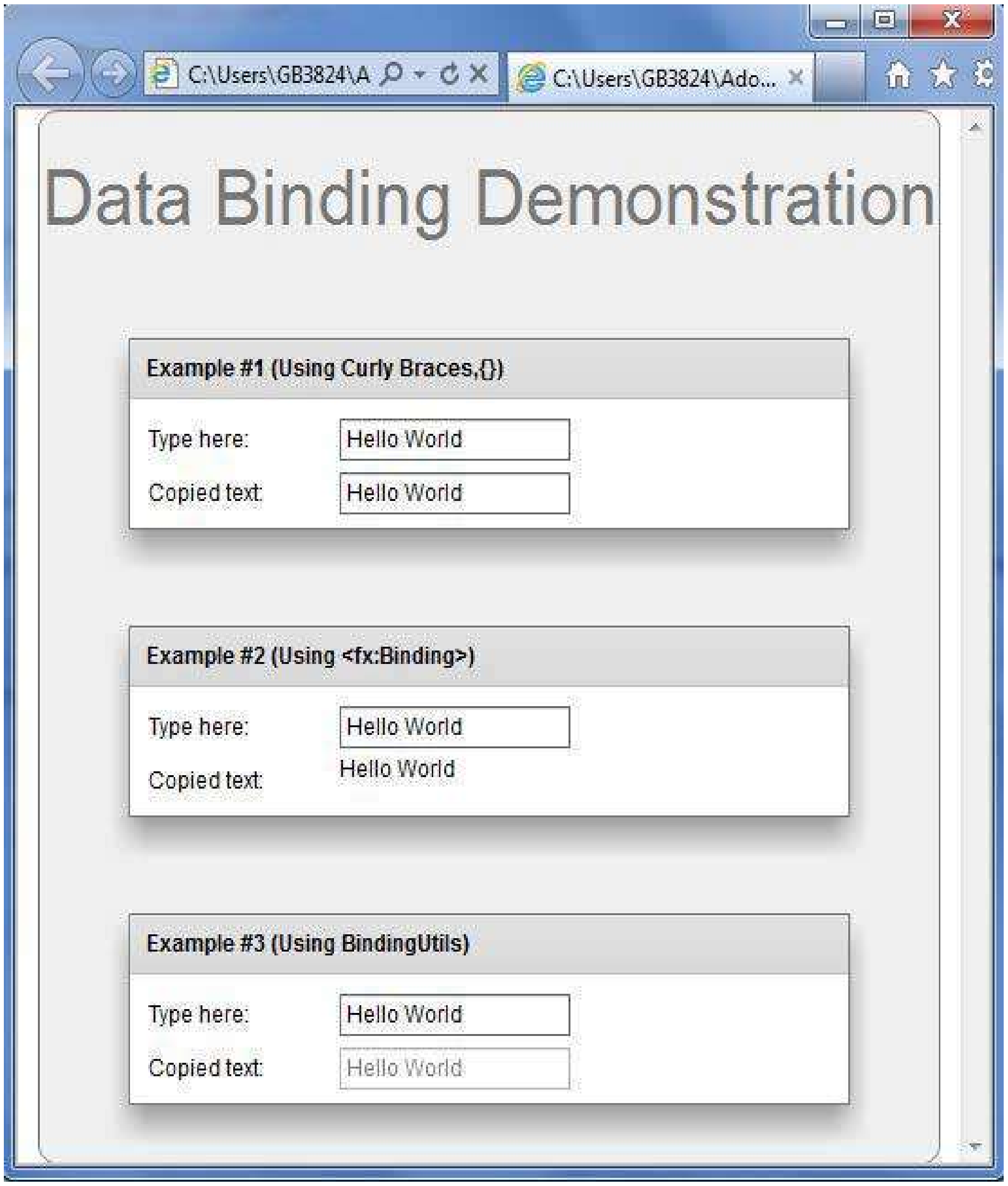
Step	Description
1	Create a project with a name *HelloWorld* under a package *com.tutorialspoint.client* as explained in the *Flex - Create Application* chapter.
2	Modify *HelloWorld.mxml* as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified **HelloWorld.mxml** file**src/com/tutorialspoint/client/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   width="100%" height="100%" minWidth="500" minHeight="500"
   >
   <fx:Style source="/com/tutorialspoint/client/Style.css"/>
   <fx:Script>
   <![CDATA[
      import mx.binding.utils.BindingUtils;
      import mx.events.FlexEvent;

      protected function txtInput6_preinitializeHandler(event:FlexEvent):void
      {
         BindingUtils.bindProperty(txtInput6,"text",txtInput5, "text");
      }
      ]]>
   </fx:Script>
   <fx:Binding source="txtInput3.text" destination="txtInput4.text" />
   <s:BorderContainer width="500" height="550" id="mainContainer"
   styleName="container">
      <s:VGroup width="100%" height="100%" gap="50" horizontalAlign="center"
      verticalAlign="middle">
```

```
         <s:Label id="lblHeader" text="Data Binding Demonstration"
         fontSize="40" color="0x777777" styleName="heading"/>
         <s:Panel title="Example #1 (Using Curly Braces,\{\})" width="400"
         height="100" >
            <s:layout>
               <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
            </s:layout>
            <s:HGroup >
               <s:Label text = "Type here: " width="100" paddingTop="6"/>
               <s:TextInput id="txtInput1"/>
            </s:HGroup>
            <s:HGroup >
               <s:Label text = "Copied text: " width="100" paddingTop="6"/>
               <s:TextInput id="txtInput2" text = "{txtInput1.text}"/>
            </s:HGroup>
         </s:Panel>
         <s:Panel title="Example #2 (Using &lt;fx:Binding&gt;)" width="400"
         height="100" >
            <s:layout>
               <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
            </s:layout>
            <s:HGroup >
               <s:Label text = "Type here: " width="100" paddingTop="6"/>
               <s:TextInput id="txtInput3"/>
            </s:HGroup>
            <s:HGroup >
               <s:Label text = "Copied text: " width="100" paddingTop="6"/>
               <s:Label id="txtInput4"/>
            </s:HGroup>
         </s:Panel>
         <s:Panel title="Example #3 (Using BindingUtils)" width="400"
            height="100" > <s:layout>
               <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
            </s:layout>
            <s:HGroup >
               <s:Label text = "Type here: " width="100" paddingTop="6"/>
               <s:TextInput id="txtInput5"/>
            </s:HGroup>
            <s:HGroup >
               <s:Label text = "Copied text: " width="100" paddingTop="6"/>
               <s:TextInput enabled="false" id="txtInput6"
               preinitialize="txtInput6_preinitializeHandler(event)"/>
            </s:HGroup>
         </s:Panel>
      </s:VGroup>
   </s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, this will produce following result: [Try it online]



Data Binding Demonstration

Example #1 (Using Curly Braces,{})

Type here: Hello World

Copied text: Hello World

Example #2 (Using <fx:Binding>)

Type here: Hello World

Copied text: Hello World

Example #3 (Using BindingUtils)

Type here: Hello World

Copied text: Hello World

CHAPTER

10

Flex – Basic Controls

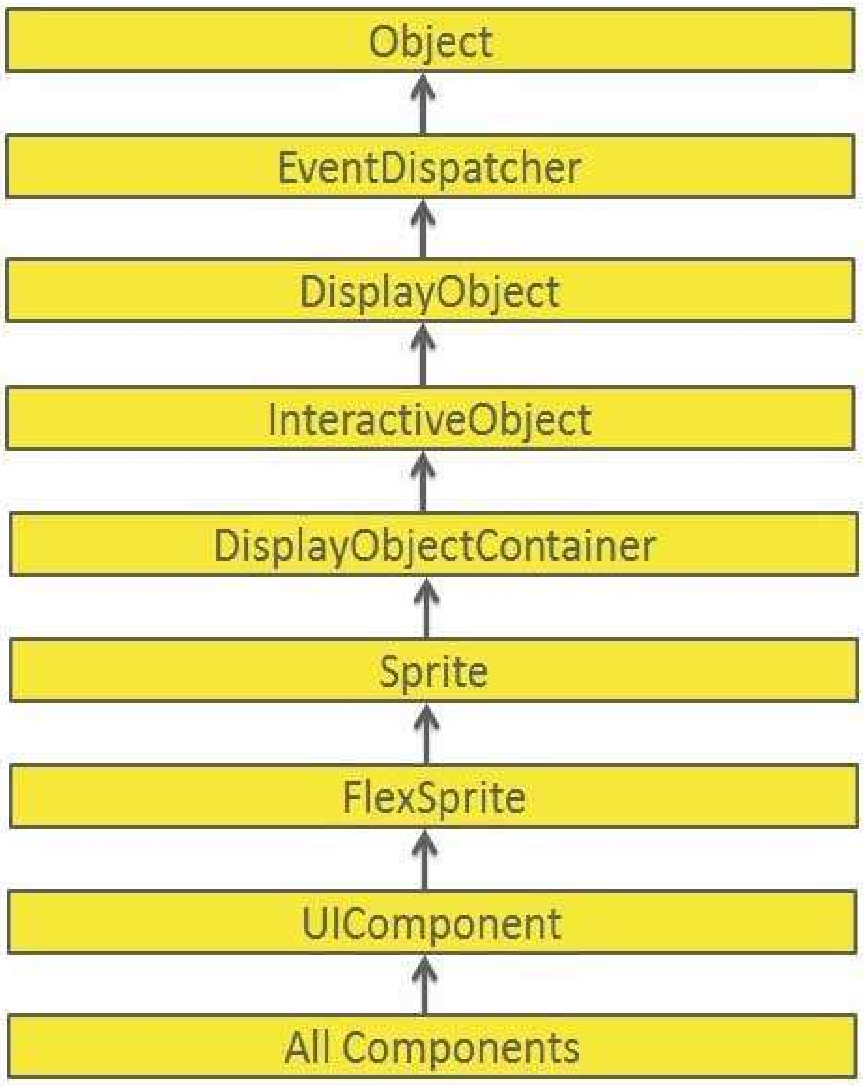
This chapter describes basic controls of Adobe Flex::.

Every user interface considers the following three main aspects:

- **UI elements** : These are the core visual elements the user eventually sees and interacts with. Flex provides a huge list of widely used and common elements varying from basic to complex which we will cover in this tutorial.
- **Layouts:** They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface). This part will be covered in Layout chapter.
- **Behavior:** These are events which occur when the user interacts with UI elements. This part will be covered in Event Handling chapter.

Flex UI Elements:

The Flex UI library provides classes in a well-defined class hierarchy to create complex web-based user interfaces. All classes in this component hierarchy has been derived from the **EventDispatcher**base class as shown below:



Every Basic UI control inherits properties from UIComponent class which in turn inherits properties from EventDispatcher and other top level classes.

S.N.	Control & Description
1	**Flex EventDispatcher Class** The EventDispatcher class is the base class for all classes that can dispatch events. The EventDispatcher class allows any object on the display list to be an event target and as such, to use the methods of the IEventDispatcher interface.
2	**Flex UIComponent** The UIComponent class is the base class for all visual components, both interactive and noninteractive.

Event Dispatcher Class

Introduction

- The **EventDispatcher** class is the base class for all classes that dispatch events.
- The **EventDispatcher** class implements the **IEventDispatcher** interface.
- The **EventDispatcher** class allows any object on the display list to be an event target and as such, to use the methods of the IEventDispatcher interface.

In order to understand **EventDispatcher**, let us first look what are event and event targets.

What is an Event?

Event is a notification when a particular action is performed.For example, when a button is clicked then Click Event occurs.

What is an Event Target

The **Event target** serves as the focal point for how events flow through the display list hierarchy.

When an event occurs, Flash Player dispatches an event object into the event flow from the root of the display list. The event object then makes its way through the display list until it reaches the event target, at which point it begins its return trip through the display list.

This round-trip journey to the event target is divided into three phases:

S.N.	Phase & Description
1	**capture** This phase comprises the journey from the root to the last node before the event target's node
2	**target** This phase comprises only the event target node.
3	**bubbling** This phase comprises any subsequent nodes encountered on the return trip to the root of the display list.

In general, any class which extends EventDispatcher gets the event dispatching capability.

Class declaration

Following is the declaration for **flash.events.EventDispatcher** class:

```
public class EventDispatcher
   extends java.lang.Object
      implements IEventDispatcher
```

Public methods

S.N.	Method & Description
1	**EventDispatcher(target:IEventDispatcher = null)** Aggregates an instance of the EventDispatcher class.
2	**addEventListener(type:String, listener:Function, useCapture:Boolean = false, priority:int = 0, useWeakReference:Boolean = false):void** Registers an event listener object with an EventDispatcher object so that the listener receives notification of an event.
3	**dispatchEvent(event:Event):Boolean** Dispatches an event into the event flow.
4	**hasEventListener(type:String):Boolean** Checks whether the EventDispatcher object has any listeners registered for a specific type of event.
5	**removeEventListener(type:String, listener:Function, useCapture:Boolean = false):void** Removes a listener from the EventDispatcher object.
6	**willTrigger(type:String):Boolean** Checks whether an event listener is registered with this EventDispatcher object or any of its ancestors for the specified event type.

Events

Following are the events for **flash.events.EventDispatcher** class:

S.N.	Event & Description
1	**activate** Dispatched when the Flash Player gains operating system focus and becomes active.
2	**detivate** Dispatched when the Flash Player loses operating system focus and becomes inactive.

Methods inherited

This class inherits methods from the following class:

- Object

UIComponent Class

Introduction

The **UIComponent** class is the base class for all visual components, both interactive and noninteractive.

Class declaration

Following is the declaration for **mx.core.UIComponent** class:

```
public class UIComponent
   extends FlexSprite
      implements IAutomationObject, IChildList, IConstraintClient,
         IDeferredInstantiationUIComponent, IFlexDisplayObject,
            IFlexModule,IInvalidating, ILayoutManagerClient,
               IPropertyChangeNotifier,IRepeaterClient, IStateClient,
                  IAdvancedStyleClient, IToolTipManagerClient,
                     IUIComponent, IValidatorListener, IVisualElement
```

Public Properties

Following are the public properties for **mx.core.UIComponent** class:

S.N.	Name & Description
1	**accessibilityDescription : String** A convenience accessor for the description property in this UIComponent's accessibilityProperties object.
2	**accessibilityEnabled : Boolean** A convenience accessor for the silent property in this UIComponent's accessibilityProperties object.
3	**accessibilityName : String** A convenience accessor for the name property in this UIComponent's accessibilityProperties object.
4	**accessibilityShortcut : String** A convenience accessor for the shortcut property in this UIComponent's accessibilityProperties object.
5	**activeEffects : Array** [read-only] The list of effects that are currently playing on the component, as an Array of EffectInstance instances.
6	**automationDelegate : Object** The delegate object that handles the automation-related functionality.
7	**automationEnabled : Boolean** [read-only] True if this component is enabled for automation, false otherwise.
8	**automationName : String** Name that can be used as an identifier for this object.
9	**automationOwner : DisplayObjectContainer** [read-only] The owner of this component for automation purposes.
10	**automationParent : DisplayObjectContainer** [read-only] The parent of this component for automation purposes.
11	**automationTabularData : Object** [read-only] An implementation of the IAutomationTabularData interface, which can be used to retrieve the data.

12	**automationValue : Array** [read-only] This value generally corresponds to the rendered appearance of the object and should be usable for correlating the identifier with the object as it appears visually within the application.
13	**automationVisible : Boolean** [read-only] True if this component is visible for automation, false otherwise.
14	**baseline : Object** For components, this layout constraint property is a facade on top of the similarly-named style.
15	**baselinePosition : Number** [read-only] The y-coordinate of the baseline of the first line of text of the component.
16	**bottom : Object** For components, this layout constraint property is a facade on top of the similarly-named style.
17	**cacheHeuristic : Boolean** [write-only] Used by Flex to suggest bitmap caching for the object.
18	**cachePolicy : String** Specifies the bitmap caching policy for this object.
19	**className : String** [read-only] The name of this instance's class, such as "Button".
20	**contentMouseX : Number** [read-only] Returns the x position of the mouse, in the content coordinate system.
21	**contentMouseY : Number** [read-only] Returns the y position of the mouse, in the content coordinate system.
22	**currentState : String** The current view state of the component.
23	**cursorManager : ICursorManager** [read-only] Gets the CursorManager that controls the cursor for this component and its peers.
24	**depth : Number** Determines the order in which items inside of containers are rendered.
25	**descriptor : UIComponentDescriptor** Reference to the UIComponentDescriptor, if any, that was used by the createComponentFromDescriptor() method to create this UIComponent instance.
26	**designLayer : DesignLayer** Specifies the optional DesignLayer instance associated with this visual element.
27	**document : Object** A reference to the document object associated with this UIComponent.
28	**doubleClickEnabled : Boolean** [override] Specifies whether the UIComponent object receives doubleClick events.
29	**enabled : Boolean** Whether the component can accept user interaction.

30	**errorString : String** The text that displayed by a component's error tip when a component is monitored by a Validator and validation fails.
31	**explicitHeight : Number** Number that specifies the explicit height of the component, in pixels, in the component's coordinates.
32	**explicitMaxHeight : Number** The maximum recommended height of the component to be considered by the parent during layout.
33	**explicitMaxWidth : Number** The maximum recommended width of the component to be considered by the parent during layout.
34	**explicitMinHeight : Number** The minimum recommended height of the component to be considered by the parent during layout.
35	**explicitMinWidth : Number** The minimum recommended width of the component to be considered by the parent during layout.
36	**explicitWidth : Number** Number that specifies the explicit width of the component, in pixels, in the component's coordinates.
37	**flexContextMenu : IFlexContextMenu** The context menu for this UIComponent.
38	**focusEnabled : Boolean** Indicates whether the component can receive focus when tabbed to.
39	**focusManager : IFocusManager** Gets the FocusManager that controls focus for this component and its peers.
40	**focusPane : Sprite** The focus pane associated with this object.
41	**hasFocusableChildren : Boolean** A flag that indicates whether child objects can receive focus.
42	**hasLayoutMatrix3D : Boolean** [read-only] Contains true if the element has 3D Matrix.
43	**height : Number** [override] Number that specifies the height of the component, in pixels, in the parent's coordinates.
44	**horizontalCenter : Object** For components, this layout constraint property is a facade on top of the similarly-named style.
45	**id : String** ID of the component.
46	**includeInLayout : Boolean** Specifies whether this component is included in the layout of the parent container.
47	**inheritingStyles : Object** The beginning of this component's chain of inheriting styles.
48	**initialized : Boolean**

	A flag that determines if an object has been through all three phases of layout: commitment, measurement, and layout (provided that any were required).
49	**instanceIndex : int** [read-only] The index of a repeated component.
50	**instanceIndices : Array** An Array containing the indices required to reference this UIComponent object from its parent document.
51	**is3D : Boolean** [read-only] Contains true when the element is in 3D.
52	**isDocument : Boolean** [read-only] Contains true if this UIComponent instance is a document object.
53	**isPopUp : Boolean** Set to true by the PopUpManager to indicate that component has been popped up.
54	**layoutMatrix3D : Matrix3D** [write-only] The transform matrix that is used to calculate a component's layout relative to its siblings.
55	**left : Object** For components, this layout constraint property is a facade on top of the similarly-named style.
56	**maintainProjectionCenter : Boolean** When true, the component keeps its projection matrix centered on the middle of its bounding box.
57	**maxHeight : Number** The maximum recommended height of the component to be considered by the parent during layout.
58	**maxWidth : Number** The maximum recommended width of the component to be considered by the parent during layout.
59	**measuredHeight : Number** The default height of the component, in pixels.
60	**measuredMinHeight : Number** The default minimum height of the component, in pixels.
61	**measuredMinWidth : Number** The default minimum width of the component, in pixels.
62	**measuredWidth : Number** The default width of the component, in pixels.
63	**minHeight : Number** The minimum recommended height of the component to be considered by the parent during layout.
64	**minWidth : Number** The minimum recommended width of the component to be considered by the parent during layout.
65	**moduleFactory : IFlexModuleFactory** A module factory is used as context for using embedded fonts and for finding the style manager that controls the styles for this component.

66	**mouseFocusEnabled : Boolean** Whether you can receive focus when clicked on.
67	**nestLevel : int** Depth of this object in the containment hierarchy.
68	**nonInheritingStyles : Object** The beginning of this component's chain of non-inheriting styles.
69	**numAutomationChildren : int** [read-only] The number of automation children this container has.
70	**owner : DisplayObjectContainer** The owner of this IVisualElement object.
71	**parent : DisplayObjectContainer** [override] [read-only] The parent container or component for this component.
72	**parentApplication : Object** [read-only] A reference to the Application object that contains this UIComponent instance.
73	**parentDocument : Object** [read-only] A reference to the parent document object for this UIComponent.
74	**percentHeight : Number** Specifies the height of a component as a percentage of its parent's size.
75	**percentWidth : Number** Specifies the width of a component as a percentage of its parent's size.
76	**postLayoutTransformOffsets : mx.geom:TransformOffsets** Defines a set of adjustments that can be applied to the object's transform in a way that is invisible to its parent's layout.
77	**processedDescriptors : Boolean** Set to true after immediate or deferred child creation, depending on which one happens.
78	**repeater : IRepeater** [read-only] A reference to the Repeater object in the parent document that produced this UIComponent.
79	**repeaterIndex : int** [read-only] The index of the item in the data provider of the Repeater that produced this UIComponent.
80	**repeaterIndices : Array** An Array containing the indices of the items in the data provider of the Repeaters in the parent document that produced this UIComponent.
81	**repeaters : Array** An Array containing references to the Repeater objects in the parent document that produced this UIComponent.
82	**right : Object** For components, this layout constraint property is a facade on top of the similarly-named style.
83	**rotation : Number** [override] Indicates the rotation of the DisplayObject instance, in degrees, from its original orientation.

84	**rotationX : Number** [override] Indicates the x-axis rotation of the DisplayObject instance, in degrees, from its original orientation relative to the 3D parent container.
85	**rotationY : Number** [override] Indicates the y-axis rotation of the DisplayObject instance, in degrees, from its original orientation relative to the 3D parent container.
86	**rotationZ : Number** [override] Indicates the z-axis rotation of the DisplayObject instance, in degrees, from its original orientation relative to the 3D parent container.
87	**scaleX : Number** [override] Number that specifies the horizontal scaling factor.
88	**scaleY : Number** [override] Number that specifies the vertical scaling factor.
89	**scaleZ : Number** [override] Number that specifies the scaling factor along the z axis.
90	**screen : Rectangle** [read-only] Returns an object that contains the size and position of the base drawing surface for this object.
91	**showInAutomationHierarchy : Boolean** A flag that determines if an automation object shows in the automation hierarchy.
92	**states : Array** The view states that are defined for this component.
93	**styleDeclaration : CSSStyleDeclaration** Storage for the inline inheriting styles on this object.
94	**styleManager : IStyleManager2** [read-only] Returns the StyleManager instance used by this component.
95	**styleName : Object** The class style used by this component.
96	**styleParent : IAdvancedStyleClient** A component's parent is used to evaluate descendant selectors.
97	**systemManager : ISystemManager** Returns the SystemManager object used by this component.
98	**tabFocusEnabled : Boolean** A flag that indicates whether this object can receive focus via the TAB key This is similar to the tabEnabled property used by the Flash Player. This is usually true for components that handle keyboard input, but some components in controlbars have them set to false because they should not steal focus from another component like an editor.
99	**toolTip : String** Text to display in the ToolTip.

100	**top : Object** For components, this layout constraint property is a facade on top of the similarly-named style.
101	**transform : flash.geom:Transform** [override] An object with properties pertaining to a display object's matrix, color transform, and pixel bounds.
102	**transformX : Number** Sets the x coordinate for the transform center of the component.
103	**transformY : Number** Sets the y coordinate for the transform center of the component.
104	**transformZ : Number** Sets the z coordinate for the transform center of the component.
105	**transitions : Array** An Array of Transition objects, where each Transition object defines a set of effects to play when a view state change occurs.
106	**tweeningProperties : Array** Array of properties that are currently being tweened on this object.
107	**uid : String** A unique identifier for the object.
108	**updateCompletePendingFlag : Boolean** A flag that determines if an object has been through all three phases of layout validation (provided that any were required).
109	**validationSubField : String** Used by a validator to associate a subfield with this component.
110	**verticalCenter : Object** For components, this layout constraint property is a facade on top of the similarly-named style.
111	**visible : Boolean** [override] Whether or not the display object is visible.
112	**width : Number** [override] Number that specifies the width of the component, in pixels, in the parent's coordinates.
113	**x : Number** [override] Number that specifies the component's horizontal position, in pixels, within its parent container.
114	**y : Number** [override] Number that specifies the component's vertical position, in pixels, within its parent container.
115	**z : Number** [override] Indicates the z coordinate position along the z-axis of the DisplayObject instance relative to the 3D parent container.

Protected Properties

Following are the protected properties for **mx.core.UIComponent** class:

S.N.	Name & Description
1	**currentCSSState : String** [read-only] The state to be used when matching CSS pseudo-selectors.
2	**hasComplexLayoutMatrix : Boolean** [read-only] Returns true if the UIComponent has any non-translation (x,y) transform properties.
3	**resourceManager : IResourceManager** [read-only] A reference to the object which manages all of the application's localized resources.
4	**unscaledHeight : Number** [read-only] A convenience method for determining the unscaled height of the component.
5	**unscaledWidth : Number** [read-only] A convenience method for determining the unscaled width of the component All of a component's drawing and child layout should be done within a bounding rectangle of this width, which is also passed as an argument to updateDisplayList().

S.N.	Event & Description
1	**activate** Dispatched when the Flash Player gains operating system focus and becomes active.
2	**detivate** Dispatched when the Flash Player loses operating system focus and becomes inactive.

Public methods

S.N.	Method & Description
1	**UIComponent()** Constructor.
2	**addStyleClient(styleClient:IAdvancedStyleClient):void** Adds a non-visual style client to this component instance.
3	**callLater(method:Function, args:Array = null):void** Queues a function to be called later.
4	**clearStyle(styleProp:String):void** Deletes a style property from this component instance.
5	**contentToGlobal(point:Point):Point** Converts a Point object from content coordinates to global coordinates.

6	**contentToLocal(point:Point):Point** Converts a Point object from content to local coordinates.
7	**createAutomationIDPart(child:IAutomationObject):Object** Returns a set of properties that identify the child within this container.
8	**createAutomationIDPartWithRequiredProperties(child:IAutomationObject, properties:Array):Object** Returns a set of properties that identify the child within this container.
9	**createReferenceOnParentDocument(parentDocument:IFlexDisplayObject):void** Creates an id reference to this IUIComponent object on its parent document object.
10	**deleteReferenceOnParentDocument(parentDocument:IFlexDisplayObject):void** Deletes the id reference to this IUIComponent object on its parent document object.
11	**determineTextFormatFromStyles():mx.core:UITextFormat** Returns a UITextFormat object corresponding to the text styles for this UIComponent.
12	**dispatchEvent(event:Event):Boolean** [override] Dispatches an event into the event flow.
13	**drawFocus(isFocused:Boolean):void** Shows or hides the focus indicator around this component.
14	**drawRoundRect(x:Number, y:Number, w:Number, h:Number, r:Object = null, c:Object = null, alpha:Object = null, rot:Object = null, gradient:String = null, ratios:Array = null, hole:Object = null):void** Programmatically draws a rectangle into this skin's Graphics object.
15	**effectFinished(effectInst:IEffectInstance):void** Called by the effect instance when it stops playing on the component.
16	**effectStarted(effectInst:IEffectInstance):void** Called by the effect instance when it starts playing on the component.
17	**endEffectsStarted():void** Ends all currently playing effects on the component.
18	**executeBindings(recurse:Boolean = false):void** Executes all the bindings for which the UIComponent object is the destination.
19	**finishPrint(obj:Object, target:IFlexDisplayObject):void** Called after printing is complete.
20	**getAutomationChildAt(index:int):IAutomationObject** Provides the automation object at the specified index.
21	**getAutomationChildren():Array** Provides the automation object list .
22	**getBoundsXAtSize(width:Number, height:Number, postLayoutTransform:Boolean = true):Number** Returns the x coordinate of the element's bounds at the specified element size.
23	**getBoundsYAtSize(width:Number, height:Number, postLayoutTransform:Boolean =**

	true):Number Returns the y coordinate of the element's bounds at the specified element size.
24	**getClassStyleDeclarations():Array** Finds the type selectors for this UIComponent instance.
25	**getConstraintValue(constraintName:String):*** Returns a layout constraint value, which is the same as getting the constraint style for this component.
26	**getExplicitOrMeasuredHeight():Number** A convenience method for determining whether to use the explicit or measured height
27	**getExplicitOrMeasuredWidth():Number** A convenience method for determining whether to use the explicit or measured width
28	**getFocus():InteractiveObject** Gets the object that currently has focus.
29	**getLayoutBoundsHeight(postLayoutTransform:Boolean = true):Number** Returns the element's layout height.
30	**getLayoutBoundsWidth(postLayoutTransform:Boolean = true):Number** Returns the element's layout width.
31	**getLayoutBoundsX(postLayoutTransform:Boolean = true):Number** Returns the x coordinate that the element uses to draw on screen.
32	**getLayoutBoundsY(postLayoutTransform:Boolean = true):Number** Returns the y coordinate that the element uses to draw on screen.
33	**getLayoutMatrix():Matrix** Returns the transform matrix that is used to calculate the component's layout relative to its siblings.
34	**getLayoutMatrix3D():Matrix3D** Returns the layout transform Matrix3D for this element.
35	**getMaxBoundsHeight(postLayoutTransform:Boolean = true):Number** Returns the element's maximum height.
36	**getMaxBoundsWidth(postLayoutTransform:Boolean = true):Number** Returns the element's maximum width.
37	**getMinBoundsHeight(postLayoutTransform:Boolean = true):Number** Returns the element's minimum height.
38	**getMinBoundsWidth(postLayoutTransform:Boolean = true):Number** Returns the element's minimum width.
39	**getPreferredBoundsHeight(postLayoutTransform:Boolean = true):Number** Returns the element's preferred height.
40	**getPreferredBoundsWidth(postLayoutTransform:Boolean = true):Number** Returns the element's preferred width.
41	**getRepeaterItem(whichRepeater:int = -1):Object** Returns the item in the dataProvider that was used by the specified Repeater to produce this Repeater,

	or null if this Repeater isn't repeated.
42	**getStyle(styleProp:String):*** Gets a style property that has been set anywhere in this component's style lookup chain.
43	**globalToContent(point:Point):Point** Converts a Point object from global to content coordinates.
45	**hasCSSState():Boolean** Returns true if currentCSSState is not null.
46	**hasState(stateName:String):Boolean** Determines whether the specified state has been defined on this UIComponent.
47	**horizontalGradientMatrix(x:Number, y:Number, width:Number, height:Number):Matrix** Returns a box Matrix which can be passed to the drawRoundRect() method as the rot parameter when drawing a horizontal gradient.
48	**initialize():void** Initializes the internal structure of this component.
49	**initializeRepeaterArrays(parent:IRepeaterClient):void** Initializes various properties which keep track of repeated instances of this component.
50	**invalidateDisplayList():void** Marks a component so that its updateDisplayList() method gets called during a later screen update.
51	**invalidateLayering():void** Called by a component's items to indicate that their depth property has changed.
52	**invalidateLayoutDirection():void** An element must call this method when its layoutDirection changes or when its parent's layoutDirection changes.
53	**invalidateProperties():void** Marks a component so that its commitProperties() method gets called during a later screen update.
54	**invalidateSize():void** Marks a component so that its measure() method gets called during a later screen update.
55	**localToContent(point:Point):Point** Converts a Point object from local to content coordinates.
56	**matchesCSSState(cssState:String):Boolean** Returns true if cssState matches currentCSSState.
57	**matchesCSSType(cssType:String):Boolean** Determines whether this instance is the same as, or is a subclass of, the given type.
58	**measureHTMLText(htmlText:String):flash.text:TextLineMetrics** Measures the specified HTML text, which can contain HTML tags such as &lt;font&> and &<b&>, assuming that it is displayed in a single-line UITextField using a UITextFormat determined by the styles of this UIComponent.
59	**measureText(text:String):flash.text:TextLineMetrics** Measures the specified text, assuming that it is displayed in a single-line UITextField (or

	UIFTETextField) using a UITextFormat determined by the styles of this UIComponent.
60	**move(x:Number, y:Number):void** Moves the component to a specified position within its parent.
61	**notifyStyleChangeInChildren(styleProp:String, recursive:Boolean):void** Propagates style changes to the children.
62	**owns(child:DisplayObject):Boolean** Returns true if the chain of owner properties points from child to this UIComponent.
63	**parentChanged(p:DisplayObjectContainer):void** Called by Flex when a UIComponent object is added to or removed from a parent.
64	**prepareToPrint(target:IFlexDisplayObject):Object** Prepares an IFlexDisplayObject for printing.
65	**regenerateStyleCache(recursive:Boolean):void** Builds or rebuilds the CSS style cache for this component and, if the recursive parameter is true, for all descendants of this component as well.
66	**registerEffects(effects:Array):void** For each effect event, registers the EffectManager as one of the event listeners.
67	**removeStyleClient(styleClient:IAdvancedStyleClient):void** Removes a non-visual style client from this component instance.
68	**replayAutomatableEvent(event:Event):Boolean** Replays the specified event.
69	**resolveAutomationIDPart(criteria:Object):Array** Resolves a child by using the id provided.
70	**resumeBackgroundProcessing():void** [static] Resumes the background processing of methods queued by callLater(), after a call to suspendBackgroundProcessing().
71	**setActualSize(w:Number, h:Number):void** Sizes the object.
72	**setConstraintValue(constraintName:String, value:*):void** Sets a layout constraint value, which is the same as setting the constraint style for this component.
73	**setCurrentState(stateName:String, playTransition:Boolean = true):void** Set the current state.
74	**setFocus():void** Sets the focus to this component.
75	**setLayoutBoundsPosition(x:Number, y:Number, postLayoutTransform:Boolean = true):void** Sets the coordinates that the element uses to draw on screen.
76	**setLayoutBoundsSize(width:Number, height:Number, postLayoutTransform:Boolean = true):void** Sets the layout size of the element.

77	**setLayoutMatrix(value:Matrix, invalidateLayout:Boolean):void** Sets the transform Matrix that is used to calculate the component's layout size and position relative to its siblings.
78	**setLayoutMatrix3D(value:Matrix3D, invalidateLayout:Boolean):void** Sets the transform Matrix3D that is used to calculate the component's layout size and position relative to its siblings.
79	**setStyle(styleProp:String, newValue:*):void** Sets a style property on this component instance.
80	**setVisible(value:Boolean, noEvent:Boolean = false):void** Called when the visible property changes.
81	**styleChanged(styleProp:String):void** Detects changes to style properties.
82	**stylesInitialized():void** Flex calls the stylesInitialized() method when the styles for a component are first initialized.
83	**suspendBackgroundProcessing():void** [static] Blocks the background processing of methods queued by callLater(), until resumeBackgroundProcessing() is called.
84	**transformAround(transformCenter:Vector3D, scale:Vector3D = null, rotation:Vector3D = null, translation:Vector3D = null, postLayoutScale:Vector3D = null, postLayoutRotation:Vector3D = null, postLayoutTranslation:Vector3D = null, invalidateLayout:Boolean = true):void** A utility method to update the rotation, scale, and translation of the transform while keeping a particular point, specified in the component's own coordinate space, fixed in the parent's coordinate space.
85	**transformPointToParent(localPosition:Vector3D, position:Vector3D, postLayoutPosition:Vector3D):void** A utility method to transform a point specified in the local coordinates of this object to its location in the object's parent's coordinates.
86	**validateDisplayList():void** Validates the position and size of children and draws other visuals.
87	**validateNow():void** Validate and update the properties and layout of this object and redraw it, if necessary.
88	**validateProperties():void** Used by layout logic to validate the properties of a component by calling the commitProperties() method.
89	**validateSize(recursive:Boolean = false):void** Validates the measured size of the component If the LayoutManager.invalidateSize() method is called with this ILayoutManagerClient, then the validateSize() method is called when it's time to do measurements.
90	**validationResultHandler(event:ValidationResultEvent):void** Handles both the valid and invalid events from a validator assigned to this component.
91	**verticalGradientMatrix(x:Number, y:Number, width:Number, height:Number):Matrix** Returns a box Matrix which can be passed to drawRoundRect() as the rot parameter when drawing a vertical gradient.

Protected method

S.N.	Method & Description
1	**adjustFocusRect(obj:DisplayObject = null):void** Adjust the focus rectangle.
2	**applyComputedMatrix():void** Commits the computed matrix built from the combination of the layout matrix and the transform offsets to the flash displayObject's transform.
3	**attachOverlay():void** This is an internal method used by the Flex framework to support the Dissolve effect.
4	**canSkipMeasurement():Boolean** Determines if the call to the measure() method can be skipped.
5	**childrenCreated():void** Performs any final processing after child objects are created.
6	**commitProperties():void** Processes the properties set on the component.
7	**createChildren():void** Create child objects of the component.
8	**createInFontContext(classObj:Class):Object** Creates a new object using a context based on the embedded font being used.
9	**createInModuleContext(moduleFactory:IFlexModuleFactory, className:String):Object** Creates the object using a given moduleFactory.
10	**dispatchPropertyChangeEvent(prop:String, oldValue:*, value:*):void** Helper method for dispatching a PropertyChangeEvent when a property is updated.
11	**focusInHandler(event:FocusEvent):void** The event handler called when a UIComponent object gets focus.
12	**focusOutHandler(event:FocusEvent):void** The event handler called when a UIComponent object loses focus.
13	**initAdvancedLayoutFeatures():void** Initializes the implementation and storage of some of the less frequently used advanced layout features of a component.
14	**initializationComplete():void** Finalizes the initialization of this component.
15	**initializeAccessibility():void** Initializes this component's accessibility code.

16	**invalidateParentSizeAndDisplayList():void** Helper method to invalidate parent size and display list if this object affects its layout (includeInLayout is true).
17	**isOurFocus(target:DisplayObject):Boolean** Typically overridden by components containing UITextField objects, where the UITextField object gets focus.
18	**keyDownHandler(event:KeyboardEvent):void** The event handler called for a keyDown event.
19	**keyUpHandler(event:KeyboardEvent):void** The event handler called for a keyUp event.
20	**measure():void** Calculates the default size, and optionally the default minimum size, of the component.
21	**resourcesChanged():void** This method is called when a UIComponent is constructed, and again whenever the ResourceManager dispatches a "change" Event to indicate that the localized resources have changed in some way.
22	**setStretchXY(stretchX:Number, stretchY:Number):void** Specifies a transform stretch factor in the horizontal and vertical direction.
23	**stateChanged(oldState:String, newState:String, recursive:Boolean):void** This method is called when a state changes to check whether state-specific styles apply to this component
24	**updateDisplayList(unscaledWidth:Number, unscaledHeight:Number):void** Draws the object and/or sizes and positions its children.

Events

Following are the events for **mx.core.UIComponent** class:

S.N.	Event & Description
1	**add** when the component is added to a container as a content child by using the addChild(), addChildAt(), addElement(), or addElementAt() method.
2	**creationComplete**

	when the component has finished its construction, property processing, measuring, layout, and drawing.
3	**currentStateChange** after the view state has changed.
4	**currentStateChanging** after the currentState property changes, but before the view state changes.
5	**dragComplete** by the drag initiator (the component that is the source of the data being dragged) when the drag operation completes, either when you drop the dragged data onto a drop target or when you end the drag-and-drop operation without performing a drop.
6	**dragDrop** by the drop target when the user releases the mouse over it.
7	**dragEnter** by a component when the user moves the mouse over the component during a drag operation.
8	**dragExit** by the component when the user drags outside the component, but does not drop the data onto the target.
9	**dragOver** by a component when the user moves the mouse while over the component during a drag operation.
10	**dragStart** by the drag initiator when starting a drag operation.
11	**effectEnd** after an effect ends.
12	**effectStart** just before an effect starts.
13	**effectStop** after an effect is stopped, which happens only by a call to stop() on the effect.
14	**enterState** after the component has entered a view state.
15	**exitState** just before the component exits a view state.
16	**hide** when an object's state changes from visible to invisible.
17	**initialize** when the component has finished its construction and has all initialization properties set.
18	**invalid** when a component is monitored by a Validator and the validation failed.
19	**mouseDownOutside** from a component opened using the PopUpManager when the user clicks outside it.

20	**mouseWheelOutside** from a component opened using the PopUpManager when the user scrolls the mouse wheel outside it.
21	**move** when the object has moved.
22	**preinitialize** at the beginning of the component initialization sequence.
23	**remove** when the component is removed from a container as a content child by using the removeChild(), removeChildAt(), removeElement(), or removeElementAt() method.
24	**resize** when the component is resized.
25	**show** when an object's state changes from invisible to visible.
26	**stateChangeComplete** after the component has entered a new state and any state transition animation to that state has finished playing.
27	**stateChangeInterrupted** when a component interrupts a transition to its current state in order to switch to a new state.
28	**toolTipCreate** by the component when it is time to create a ToolTip.
29	**toolTipEnd** by the component when its ToolTip has been hidden and is to be discarded soon.
30	**toolTipHide** by the component when its ToolTip is about to be hidden.
31	**toolTipShow** by the component when its ToolTip is about to be shown.
32	**toolTipShown** by the component when its ToolTip has been shown.
33	**toolTipStart** by a component whose toolTip property is set, as soon as the user moves the mouse over it.
34	**touchInteractionEnd** A non-cancellable event, by a component when it is done responding to a touch interaction user gesture.

35	**touchInteractionStart** A non-cancellable event, by a component when it starts responding to a touch interaction user gesture.
36	**touchInteractionStarting** A cancellable event, by a component in an attempt to respond to a touch interaction user gesture.
37	**updateComplete** when an object has had its commitProperties(), measure(), and updateDisplayList() methods called (if needed).
38	**valid** when a component is monitored by a Validator and the validation succeeded.
39	**valueCommit** when values are changed programmatically or by user interaction.

Methods inherited

This class inherits methods from the following classes:

- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

Basic Controls

Following are few important *Basic Controls*: (Use Ctrl + Click to follow the link)

S.N.	Controls & Description
1	Label Label is a low-level UIComponent that can render one or more lines of uniformly-formatted text.
2	Text The Text control lets you display HTML content as well as normal text in your application.

3	Image The Image control lets you import JPEG, PNG, GIF, and SWF files at runtime.
4	LinkButton The LinkButton control is a borderless Button control whose contents are highlighted when a user moves the mouse over it.

CHAPTER

11

Flex – Form Controls

This chapter describes the form controls under Adobe Flex::.

Form controls allows users to input data and provides them interaction capability with the application. Every Form UI control inherits properties from UIComponent class which in turn inherits properties from EventDispatcher and other top level classes.

(Use Ctrl + Click to follow the link)

S.N.	Control & Description
1	Flex EventDispatcher Class The EventDispatcher class is the base class for all classes that can dispatch events. The EventDispatcher class allows any object on the display list to be an event target and as such, to use the methods of the IEventDispatcher interface.
2	Flex UIComponent The UIComponent class is the base class for all visual components, both interactive and noninteractive.

Form Controls

Following are few important Form Controls: (Use Ctrl + Click to follow the link)

S.N.	Control & Description
1	Button The Button component is a commonly used rectangular button.
2	ToggleButton The ToggleButton component defines a toggle button.
3	CheckBox The CheckBox component consists of an optional label and a small box that can contain a check mark or not.
4	ColorPicker The ColorPicker control provides a way for a user to choose a color from a swatch list.
5	ComboBox

	The ComboBox control is a child class of the DropDownListBase control.
6	DateChooser The DateChooser control displays the name of a month, the year, and a grid of the days of the month, with columns labeled for the day of the week.
7	RadioButton The RadioButton component allows the user make a single choice within a set of mutually exclusive choices.
8	TextArea TextArea is a text-entry control that lets users enter and edit multiple lines of formatted text.
9	TextInput TextInput is a text-entry control that lets users enter and edit a single line of uniformly-formatted text.
10	DropDownList The DropDownList control contains a drop-down list from which the user can select a single value.
11	NumericStepper The NumericStepper control lets you select a number from an ordered set.

CHAPTER

12

Flex – Complex Controls

This chapter describes the complex controls under Adobe Flex::.

Complex controls provides users advanced capabilities to deal with large amount of data in an easy way and provides them interaction capability with the application. Every Complex UI control inherits properties from UIComponent class which in turn inherits properties from EventDispatcher and other top level classes.

(Use Ctrl + Click to follow the link)

S.N.	Control & Description
1	Flex EventDispatcher Class The EventDispatcher class is the base class for all classes that can dispatch events. The EventDispatcher class allows any object on the display list to be an event target and as such, to use the methods of the IEventDispatcher interface.
2	Flex UIComponent The UIComponent class is the base class for all visual components, both interactive and noninteractive.

Complex Controls

Following are few important *Complex Controls* (Use Ctrl + Click to follow the link)

S.N.	Control & Description
1	DataGrid The DataGrid control displays a row of column headings above a scrollable grid.
2	AdvancedDataGrid The AdvancedDataGrid adds few additional functionality to the standard DataGrid control to add data visualization features.
3	Menu The Menu control creates a pop-up menu of individually selectable choices.
4	ProgressBar The ProgressBar control provides a visual representation of the progress of a task over time.

5	RichTextEditor The RichTextEditor control lets users enter and format text.
6	TileList The TileList control The TileList control displays a number of items laid out in tiles.
7	Tree The Tree control lets a user view hierarchical data arranged as an expandable tree.
8	VideoPlayer The VideoPlayer control is a skinnable video player that supports progressive download, multi-bitrate streaming, and streaming video.
9	Accordian An Accordian control has a collection of child MX containers or Spark NavigatorContent containers, but only one of them at a time is visible.
10	TabNavigator The TabNavigator control includes a TabBar container for navigating between its child containers.
11	ToggleButtonBar The ToggleButtonBar control defines a horizontal or vertical group of buttons that maintain their selected or deselected state.

CHAPTER

13

Flex – Layout Panels

This chapter describes layout panels under Adobe Flex:

Layout panel controls provides users to organize UI controls on the page. Every Layout control inherits properties from UIComponent class which in turn inherits properties from EventDispatcher and other top level classes.

(Use Ctrl + Click to follow the link)

S.N.	Control & Description
1	Flex EventDispatcher Class The EventDispatcher class is the base class for all classes that can dispatch events. The EventDispatcher class allows any object on the display list to be an event target and as such, to use the methods of the IEventDispatcher interface.
2	Flex UIComponent The UIComponent class is the base class for all visual components, both interactive and noninteractive.

Layout Panels

Following are few important *Layout Panels*: (Use Ctrl + Click to follow the link)

S.N.	Panel & Description
1	BorderContainer The BorderContainer class provides a set of CSS styles that control the appearance of the border and background fill of the container.
2	Form The Form container provides control over the layout of a form, mark form fields as required or optional, handle error messages, and bind form data to the Flex data model to perform data checking and validation.
3	VGroup The VGroup container is a Group container that uses the VerticalLayout class.

4	HGroup The HGroup container is a Group container that uses the HorizontalLayout class.
5	Panel The Panel class is a container that includes a title bar, a caption, a border, and a content area for its children.
6	SkinnableContainer The SkinnableContainer class is the base class for skinnable containers that provide visual content.
7	TabBar The TabBar displays a set of identical tabs.
8	TitleWindow The TitleWindow extends Panel to include a close button and move area.

CHAPTER

14

Flex – Visual Effects

This chapter describes the visual effects to add behavior to Flex application:

We can add behaviour to flex application using concept of Effects. For example, when a text box get focus, we can make its text become bolder and make its size slight bigger.

Every Effect inherits properties from Effect class which in turn inherits properties from EventDispatcher and other top level classes.

S.N.	Effect & Description
1	**Flex Effect Class** The Effect class is an abstract base class that defines the basic functionality of all Flex effects. This class defines the base factory class for all effects.

FlexEffect Class

Introduction

The Effect class is an abstract base class that defines the basic functionality of all Flex effects. The Effect class defines the base factory class for all effects. The Effec tInstance class defines the base class for all effect instance subclasses.

Class declaration

Following is the declaration for **mx.effects.Effect** class:

```
public class Effect
   extends EventDispatcher
      implements IEffect
```

Public properties

S.N.	Property & Description

1	**className : String** [read-only] The name of the effect class, such as "Fade".
2	**customFilter : mx.effects:EffectTargetFilter** Specifies a custom filter object, of type EffectTargetFilter, used by the effect to determine the targets on which to play the effect.
3	**duration : Number** Duration of the effect in milliseconds.
4	**effectTargetHost : mx.effects:IEffectTargetHost** A property that lets you access the target list-based control of a data effect.
5	**filter : String** Specifies an algorithm for filtering targets for an effect.
6	**hideFocusRing : Boolean** Determines whether the effect should hide the focus ring when starting the effect.
7	**instanceClass : Class** An object of type Class that specifies the effect instance class class for this effect class.
8	**isPlaying : Boolean** [read-only] A read-only flag which is true if any instances of the effect are currently playing, and false if none are.
9	**perElementOffset : Number** Additional delay, in milliseconds, for effect targets after the first target of the effect.
10	**playheadTime : Number** Current time position of the effect.
11	**relevantProperties : Array** An Array of property names to use when performing filtering.
12	**relevantStyles : Array** An Array of style names to use when performing filtering.
13	**repeatCount : int = 1** Number of times to repeat the effect.
14	**repeatDelay : int = 0** Amount of time, in milliseconds, to wait before repeating the effect.
15	**startDelay : int = 0** Amount of time, in milliseconds, to wait before starting the effect.
16	**suspendBackgroundProcessing : Boolean = false** If true, blocks all background processing while the effect is playing.
17	**target : Object** The object to which this effect is applied.
18	**targets : Array** An Array of objects that are targets for the effect.

19	**triggerEvent : Event** The Event object passed to this Effect by the EffectManager when an effect is triggered, or null if the effect is not being played by the EffectManager.

Protected properties

S.N.	Property & Description
1	**applyTransitionEndProperties : Boolean** This flag controls whether the effect, when run in a transition, automatically applies the property values according to the end state, as opposed to leaving values as set by the effect itself.
2	**endValuesCaptured : Boolean = false** A flag containing true if the end values of an effect have already been determined, or false if they should be acquired from the current properties of the effect targets when the effect runs.

Public methods

S.N.	Method & Description
1	**Effect(target:Object = null)** Constructor.
2	**captureEndValues():void** Captures the current values of the relevant properties on the effect's targets and saves them as end values.
3	**captureMoreStartValues(targets:Array):void** Captures the current values of the relevant properties of an additional set of targets Flex uses this function when a data change effect is run.
4	**captureStartValues():void** Captures the current values of the relevant properties on the effect's targets.
5	**createInstance(target:Object = null):IEffectInstance** Creates a single effect instance and initializes it.
6	**createInstances(targets:Array = null):Array** Takes an Array of target objects and invokes the createInstance() method on each target.
7	**deleteInstance(instance:IEffectInstance):void** Removes event listeners from an instance and removes it from the list of instances.
8	**end(effectInstance:IEffectInstance = null):void** Interrupts an effect that is currently playing, and jumps immediately to the end of the effect.
9	**getAffectedProperties():Array** Returns an Array of Strings, where each String is the name of a property changed by this effect.
10	**pause():void** Pauses the effect until you call the resume() method.

11	**play(targets:Array = null, playReversedFromEnd:Boolean = false):Array** Begins playing the effect.
12	**resume():void** Resumes the effect after it has been paused by a call to the pause() method.
13	**reverse():void** Plays the effect in reverse, if the effect is currently playing, starting from the current position of the effect.
14	**stop():void** Stops the effect, leaving the effect targets in their current state.

Protected methods

S.N.	Method & Description
1	**applyValueToTarget(target:Object, property:String, value:*, props:Object):void** Used internally by the Effect infrastructure.
2	**effectEndHandler(event:EffectEvent):void** Called when an effect instance has finished playing.
3	**effectStartHandler(event:EffectEvent):void** This method is called when the effect instance starts playing.
4	**effectStopHandler(event:EffectEvent):void** Called when an effect instance has stopped by a call to the stop() method.
5	**filterInstance(propChanges:Array, target:Object):Boolean** Determines the logic for filtering out an effect instance.
6	**getValueFromTarget(target:Object, property:String):*** Called by the captureStartValues() method to get the value of a property from the target.
7	**initInstance(instance:IEffectInstance):void** Copies properties of the effect to the effect instance.

Events

S.N.	Event & Description
1	**effectEnd** Dispatched when one of the effect's instances finishes playing, either when the instance finishes playing or when the effect is interrupted by a call to the end() method.

2	**effectStart** Dispatched when the effect starts playing.
3	**effectStop** Dispatched when the effect has been stopped, which only occurs when the effect is interrupted by a call to the stop() method.

Methods inherited

This class inherits methods from the following classes:

- flash.events.EventDispatcher
- Object

Basic Effects

Following are few important *Basic Visual Effects*:

S.N.	Effect & Description
1	Fade The Fade effect animates the alpha property of a component.
2	WipeLeft The WipeLeft class defines a wipe left effect.
3	WipeRight The WipeRight class defines a wipe right effect.
4	Move3D The Move3D class moves a target object in the x, y, and z dimensions.
5	Scale3D The Scale3D class scales a target object in three dimensions around the transform center.
6	Rotate3D The Rotate3D class rotate a target object in three dimensions around the x, y, or z axes.
7	Animate This Animate effect animates an arbitrary set of properties between values. Specify the properties and values to animate by setting the motionPaths property.

CHAPTER

15

Flex – Event Handling

This chapter describes the usage of Event Handling:

Flex uses concept of event to pass data from one object to another depend upon the state or user interaction within the application.

ActionScript has a generic **Event** class which defines much of the functionality needed to work with events. Every time an event occurs within a Flex application, three types of objects from the Event class hiearchy are created.

Event has the following three key properties

Property	Description
type	type states about what kind of event just happened. This may be click, initialize, mouseover, change, etc. The actual values will be represented by constants like MouseEvent.CLICK.
target	The target property of Event is an object reference to the component that generated the event.If you click a Button with an id of clickMeButton, the target of that click event will be clickMeButton
currentTarget	The currentTarget property varies container hiearchy. It mainly deals with flow of events.

Event Flow Phases

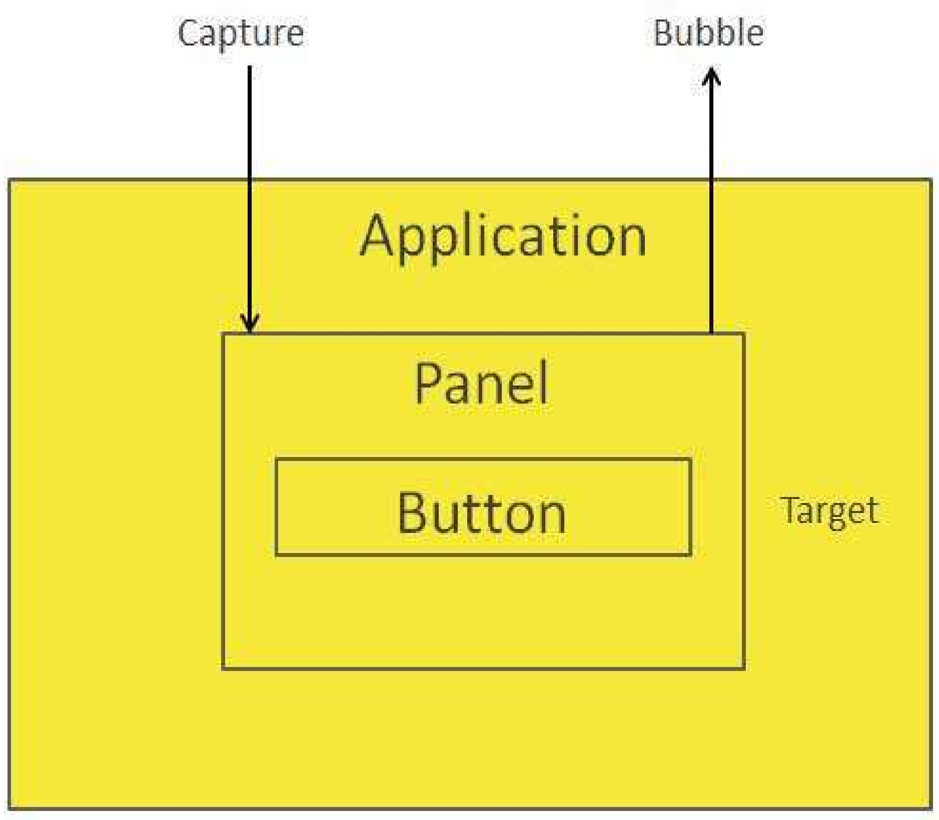
An event goes through three phases looking for event handlers.

Phase	Description
Capture	In the capture phase the program will start looking for event handlers from the outside (or top) parent to the innermost one. The capture phase stops at the parent of the object that triggered the event.
Target	In the target phase ,the component that triggered the event, is checked for an event handler.
Bubble	The Bubble phase is reverse of capture phase, working back through the structure, from the target component's parent on up.

Consider the following application code

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   width="100%" height="100%"
   minWidth="500" minHeight="500" >
   <s:Panel>
      <s:Button id="clickMeButton" label="Click Me!" click="doAction( );"/>
   </s:Panel>
</s:Application>
```

When the user clicks the Button, he or she has also clicked the Panel and the Application.The event goes through three phases looking for event-handler assignments.



Let us follow the following steps to test event handing in a Flex application:

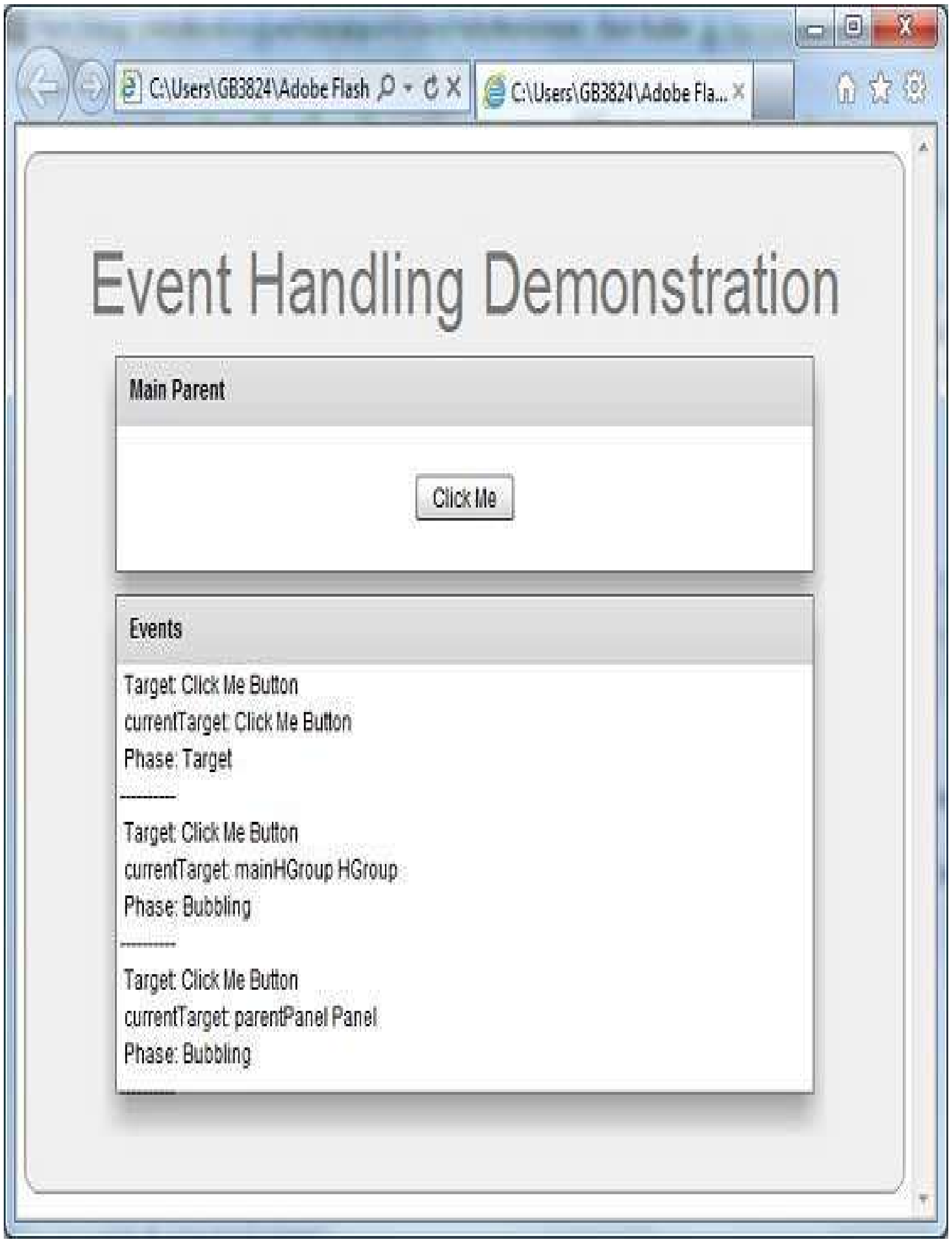
Step	Description
1	Create a project with a name *HelloWorld* under a package *com.tutorialspoint.client* as explained in the *Flex - Create Application* chapter.
2	Modify *HelloWorld.mxml* as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   width="100%" height="100%" minWidth="500" minHeight="500"
   >
   <fx:Style source="/com/tutorialspoint/client/Style.css"/>
   <fx:Script>
      <![CDATA[
         protected function reportEvent(event:MouseEvent):void
         {
            var target:String = event.target.id;
            var currentTarget:String = event.target.id;
            var eventPhase: String;

            if(event.target is Button){
               var button:Button = event.target as Button;
               target = button.label + " Button";
            } else if(event.target is HGroup){
               var hGroup:HGroup = event.target as HGroup;
               target = hGroup.id + " HGroup";
            }else if(event.target is Panel){
               var panel:Panel = event.target as Panel;
               target = panel.id + " Panel";
            }

            if(event.currentTarget is Button){
               var button1:Button = event.currentTarget as Button;
               currentTarget = button1.label + " Button";
            }else if(event.currentTarget is HGroup){
               var hGroup1:HGroup = event.currentTarget as HGroup;
               currentTarget = hGroup1.id + " HGroup";
            }else if(event.currentTarget is Panel){
               var panel1:Panel = event.currentTarget as Panel;
               currentTarget = panel1.id + " Panel";
            }

            var eventPhaseInt:uint = event.eventPhase;

            if(eventPhaseInt == EventPhase.AT_TARGET){
               eventPhase = "Target";
            } else if(eventPhaseInt == EventPhase.BUBBLING_PHASE){
               eventPhase = "Bubbling";
            }else if(eventPhaseInt == EventPhase.CAPTURING_PHASE){
               eventPhase = "Capturing";
            }

            reports.text += " Target: " + target + "\n currentTarget: " +
            currentTarget + "\n Phase: " + eventPhase + "\n----------\n";
         }
      ]]>
   </fx:Script>
   <s:BorderContainer width="630" height="480" id="mainContainer"
      styleName="container">
      <s:VGroup width="100%" height="100%" gap="10"
         horizontalAlign="center" verticalAlign="middle">
```

```
          <s:Label id="lblHeader" text="Event Handling Demonstration"
             fontSize="40" color="0x777777" styleName="heading"/>
          <s:Panel id="parentPanel" title="Main Parent"
             click="reportEvent(event)" width="500"
             height="100" includeInLayout="true" visible="true">
             <s:layout>
                <s:VerticalLayout  gap="10"
                   verticalAlign="middle" horizontalAlign="center"/>
             </s:layout>
             <s:HGroup id="mainHGroup" click="reportEvent(event)">
                <s:Button label="Click Me"
                   click="reportEvent(event)"/>
             </s:HGroup>
          </s:Panel>
          <s:Panel id="reportPanel"
             title="Events" width="500" height="230">
             <mx:Text id="reports" />
          </s:Panel>
       </s:VGroup>
    </s:BorderContainer>
 </s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, this will produce following result: [Try it online]


C:\Users\GB3824\Adobe Flash
C:\Users\GB3824\Adobe Fla...
Event Handling Demonstration
Main Parent
Click Me
Events

```
Target: Click Me Button
currentTarget: Click Me Button
Phase: Target
----------
Target: Click Me Button
currentTarget: mainHGroup HGroup
Phase: Bubbling
----------
Target: Click Me Button
currentTarget: parentPanel Panel
Phase: Bubbling
----------
```

CHAPTER

16

Flex – Custom Controls

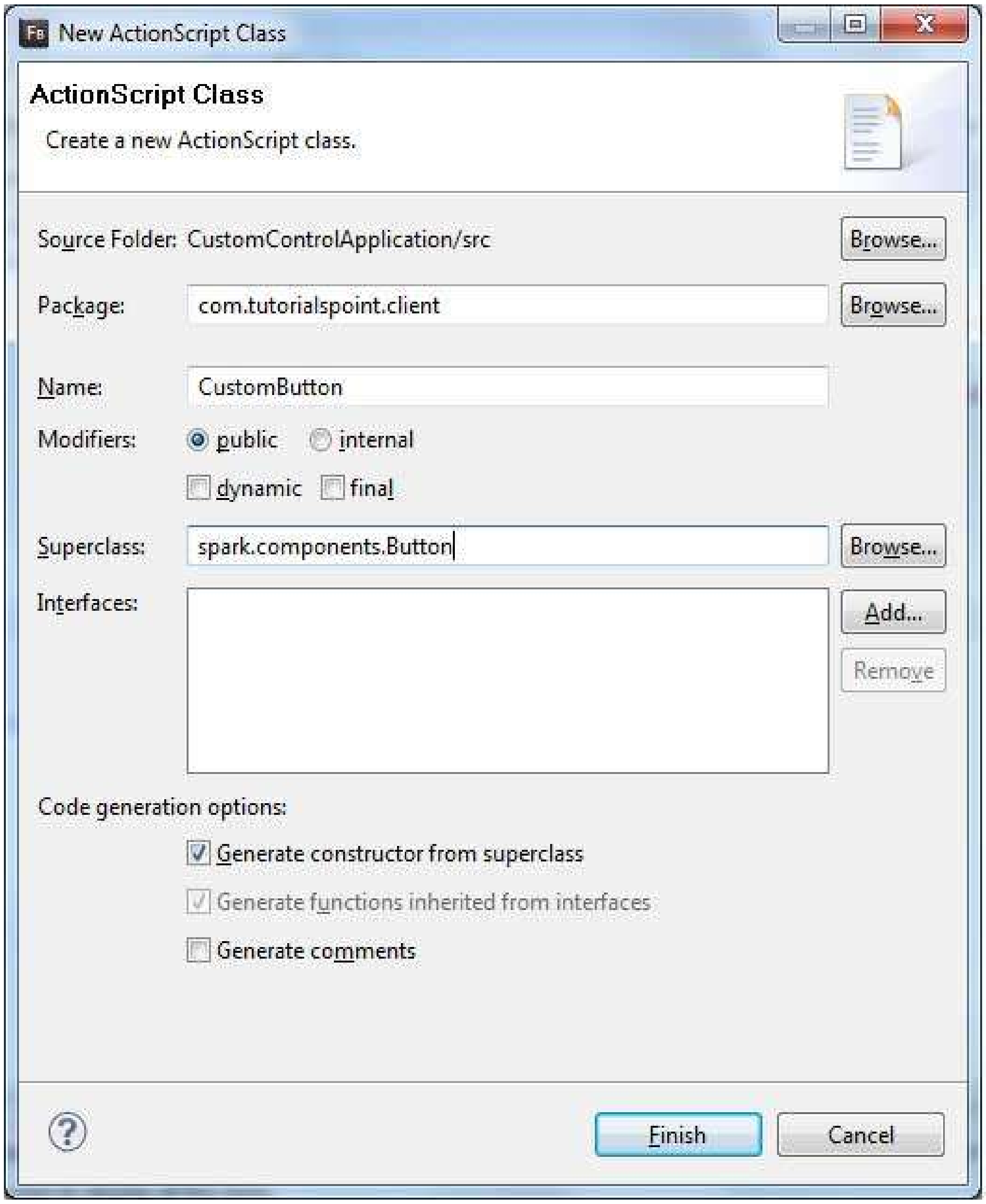
This chapter describes the Custom Controls under Adobe Flex:.

Flex provides two ways to create custom components.

- Using ActionScript
- Using MXML

Using ActionScript

You can create a component by extending existing component. To create a component using Flash Builder, Click on ***File > New > ActionScript Class***. Enter the details as shown below



ActionScript Class

Create a new ActionScript class.

Source Folder: CustomControlApplication/src

Package: com.tutorialspoint.client

Name: CustomButton

Modifiers: [✓ public] internal

☐ dynamic ☐ final

Superclass: spark.components.Button

Interfaces:

Code generation options:

☑ Generate constructor from superclass

☑ Generate functions inherited from interfaces

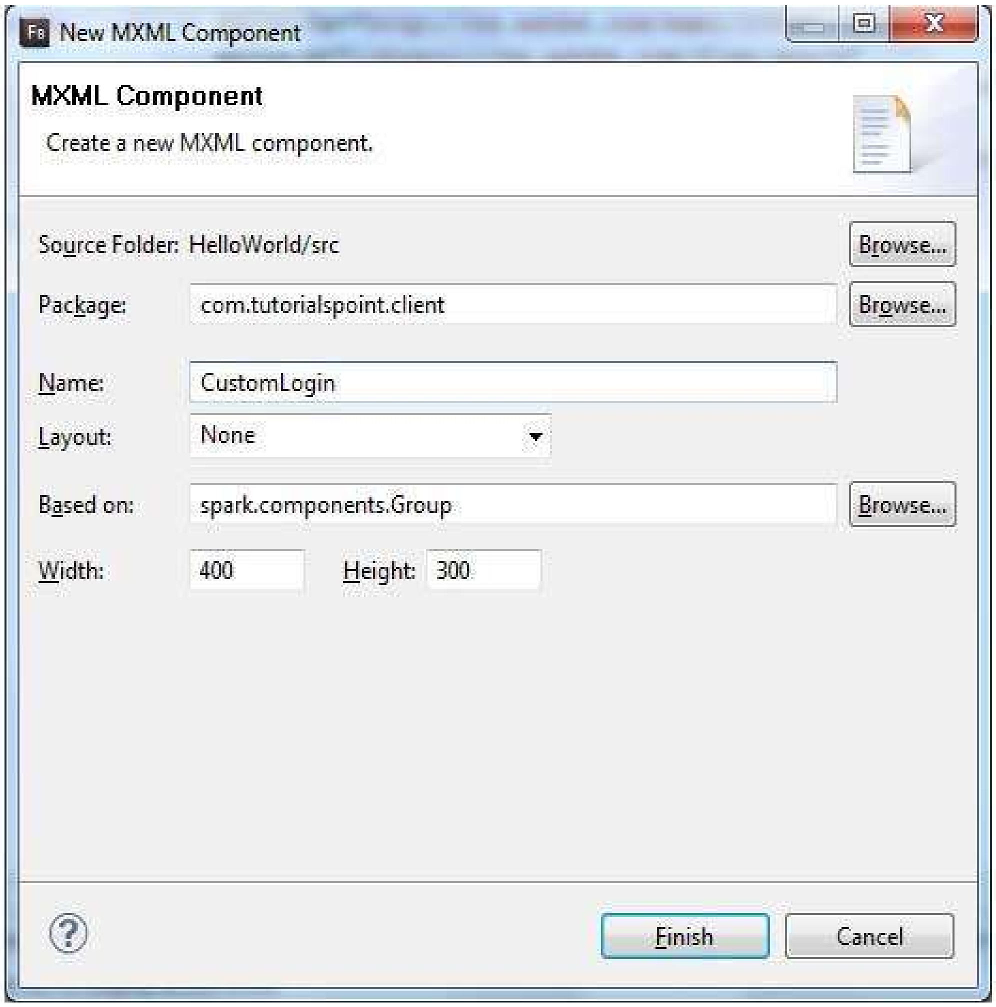
☐ Generate comments

Flash Builder will create the following CustomButton.as file.

```
package com.tutorialspoint.client
{
   import spark.components.Button;

   public class CustomButton extends Button
   {
      public function CustomButton()
      {
         super();
      }
   }
}
```

Using MXML

You can create a component by extending existing component. To create a component using Flash Builder, Click on ***File > New > MXML Component***. Enter the details as shown below.



Source Folder: HelloWorld/src
Package: com.tutorialspoint.client
Name: CustomLogin
Layout: None
Based on: spark.components.Group
Width: 400 Height: 300

Flash Builder will create the following CustomLogin.mxml file.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Group xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   width="400" height="300">

</s:Group>
```

Let us follow the following steps to test custom controls in a Flex application:

Step	Description
1	Create a project with a name *HelloWorld* under a package *com.tutorialspoint.client* as explained in the *Flex - Create Application* chapter.
2	Modify *HelloWorld.mxml* as explained below. Keep rest of the files unchanged.
4	Create *CustomLogin.mxml* and *CustomButton.as* component as explained above. Modify these files as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/client/CustomLogin.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Group xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx" width="400" height="300">
   <s:Form>
      <s:FormItem label="UserName:">
         <s:TextInput width="200" />
      </s:FormItem>
      <s:FormItem label="Password:">
         <s:TextInput width="200"
            displayAsPassword="true" />
      </s:FormItem>
      <s:FormItem>
         <s:Button label="Login" />
      </s:FormItem>
   </s:Form>
</s:Group>
```

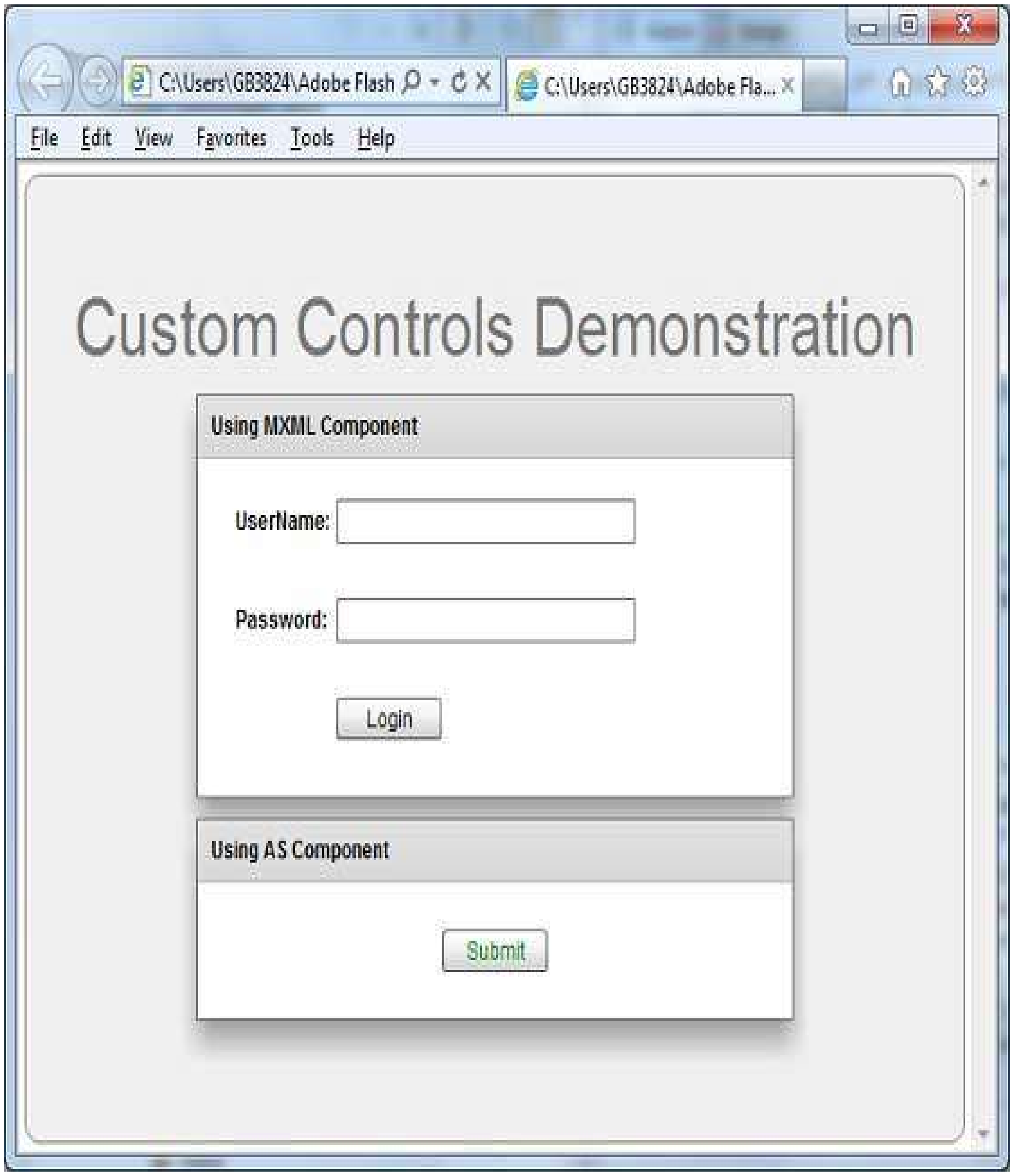
Following is the content of the modified mxml file **src/com.tutorialspoint/client/CustomButton.as**.

```
package com.tutorialspoint.client
{
   import spark.components.Button;

   public class CustomButton extends Button
   {
      public function CustomButton()
      {
         super();
         this.setStyle("color","green");
         this.label = "Submit";
      }
   }
}
```

Following is the content of the modified mxml file **src/com.tutorialspoint/client/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   xmlns:client="com.tutorialspoint.client.*"
   initialize="application_initializeHandler(event)"
   >
   <fx:Style source="/com/tutorialspoint/client/Style.css"/>
   <fx:Script>
      <![CDATA[
        import mx.events.FlexEvent;

        protected function application_initializeHandler(event:FlexEvent):void
        {
           //create a new custom button
           var customButton: CustomButton = new CustomButton();
           asPanel.addElement(customButton);
        }

      ]]>
   </fx:Script>
   <s:BorderContainer width="630" height="480" id="mainContainer"
      styleName="container">
      <s:VGroup width="100%" height="100%" gap="10"
         horizontalAlign="center" verticalAlign="middle">
         <s:Label id="lblHeader" text="Custom Controls Demonstration"
            fontSize="40" color="0x777777" styleName="heading"/>

         <s:Panel title="Using MXML Component" width="400" height="200">
            <client:CustomLogin>
            </client:CustomLogin>
         </s:Panel>
         <s:Panel  title="Using AS Component" width="400" height="100">
            <s:VGroup id="asPanel" width="100%" height="100%" gap="10"
               horizontalAlign="center" verticalAlign="middle">
            </s:VGroup>
         </s:Panel>
      </s:VGroup>
   </s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, this will produce following result: [Try it online]


C:\Users\GB3824\Adobe Flash
C:\Users\GB3824\Adobe Fla...
File
Edit
View
Favorites
Tools
Help
Custom Controls Demonstration
Using MXML Component
UserName:
Password:
Login
Using AS Component
Submit

CHAPTER

17

Flex – RPC Services

This chapter describes RPC Services provided by Adobe Flex::.

Flex provides RPC services to provide server side data to client side. Flex provides a fair amount of controll on to server side data.

- Using Flex RPC services, we can define user actions to be executed on server side.
- Flex RPC Sservices can be integrated with any server side technologies.
- One of Flex RPC Service provide inbuilt support for compressed binary data to be transferred over the wire and is pretty fast.

Flex provides following three types of RPC Services

RPC Service	Description
HttpService	<mx:HTTPService> tag is used to represent an HTTPService object in an MXML file. When you make a call to HTTPService object's send() method, it makes an HTTP request to the specified URL, and an HTTP response is returned.You can also use the HTTP HEAD, OPTIONS, TRACE, and DELETE methods.
WebService	The <mx:WebService> tag is used to get access to the operations of SOAP-compliant web services.
RemoteObject	The <mx:RemoteObject> tag is used to represent an HTTPService object in an MXML file. This tag gives you access to the methods of Java objects using Action Message Format (AMF) encoding.

We're going to discuss HTTP Service in detail. We'll use an XML source file placed at server and access it at client side via HTTP Service

Items.xml

```
<items>
   <item name="Book" description="History of France"></item>
   <item name="Pen" description="Parker Pen"></item>
   <item name="Pencil" description="Stationary"></item>
<items>
```

HTTPService Declaration

Now declare a HTTPService and pass it url of the above file

```
<fx:Declarations>
   <mx:HTTPService id="itemRequest"
   url="http://www.tutorialspoint.com/flex/Items.xml" />
</fx:Declarations>
```

RPC Call

Make a call to itemRequest.send() method and bind values from lastResult object of itemRequest webservice to Flex UI component.

```
...
itemRequest.send();
...
<mx:DataGrid id="dgItems" height="80%" width="75%"
   dataProvider="{itemRequest.lastResult.items.item}">
   <mx:columns>
      <mx:DataGridColumn headerText="Name" dataField="name"/>
      <mx:DataGridColumn headerText="Description" dataField="description"/>
   </mx:columns>
</mx:DataGrid>
```

RPC Service Call Example

Now Let us follow the following steps to test RPC services in a Flex application:

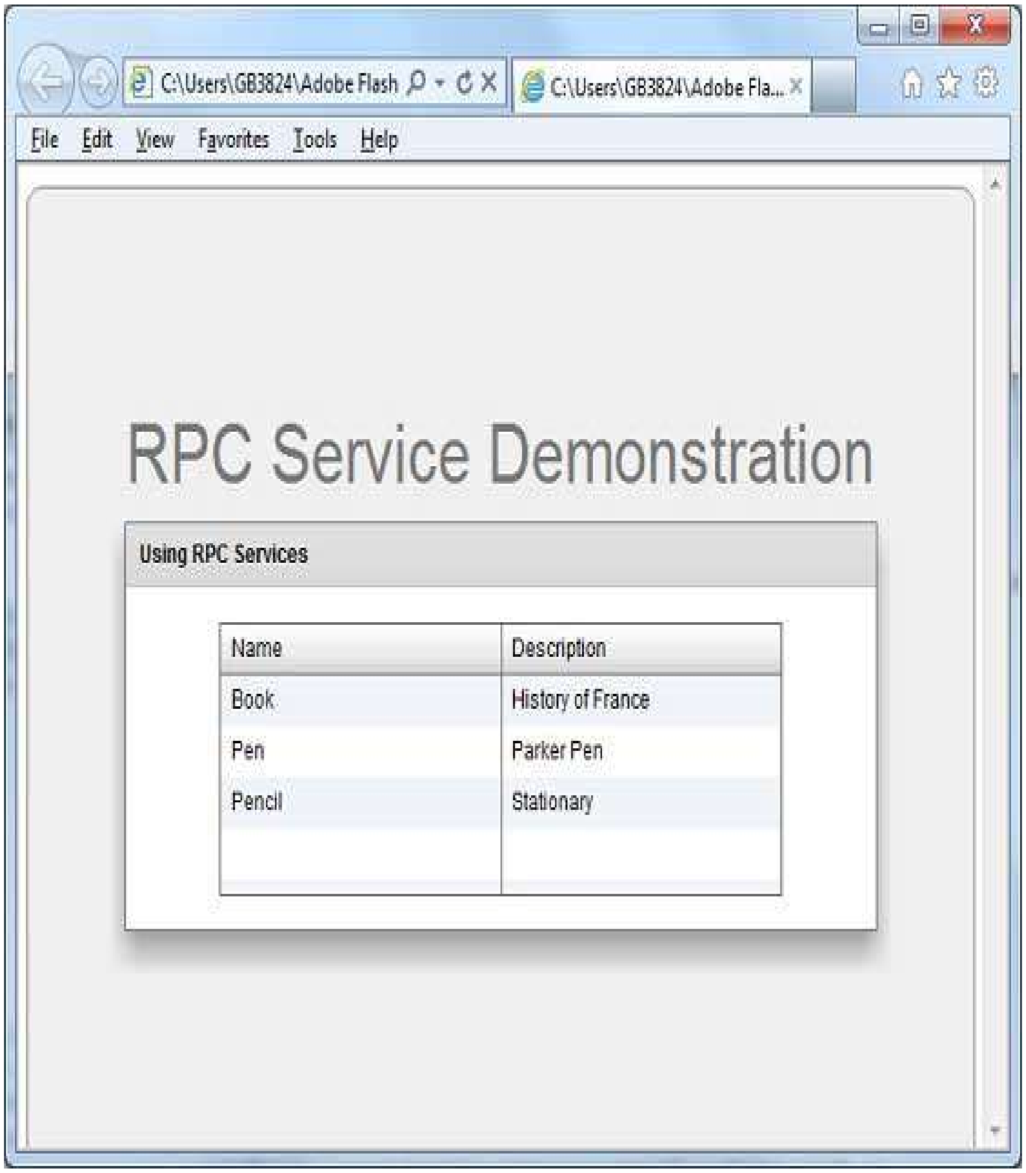
S.N.	Description
1	Create a project with a **name** *HelloWorld* under a package *com.tutorialspoint.client* as explained in the *Flex - Create Application* chapter.
2	Modify *HelloWorld.mxml* as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   minWidth="500" minHeight="500" creationComplete="init(event)">
   <fx:Style source="/com/tutorialspoint/client/Style.css"/>
   <fx:Script>
      <![CDATA[
         import mx.events.FlexEvent;
         import mx.rpc.events.FaultEvent;
         import mx.rpc.events.ResultEvent;

         protected function init(event:FlexEvent):void
         {
            itemRequest.send();
         }

      ]]>
   </fx:Script>
   <fx:Declarations>
      <mx:HTTPService id="itemRequest"
      url="http://www.tutorialspoint.com/flex/Items.xml" />
   </fx:Declarations>
   <s:BorderContainer width="630" height="480" id="mainContainer"
      styleName="container">
      <s:VGroup width="100%" height="100%" gap="10"
         horizontalAlign="center" verticalAlign="middle">
         <s:Label id="lblHeader" text="RPC Service Demonstration"
            fontSize="40" color="0x777777" styleName="heading"/>
         <s:Panel id="parentPanel" title="Using RPC Services"
            width="500" height="200" >
            <s:layout>
               <s:VerticalLayout  gap="10"
                  verticalAlign="middle" horizontalAlign="center"/>
            </s:layout>
            <mx:DataGrid id="dgItems" height="80%" width="75%"
               dataProvider="{itemRequest.lastResult.items.item}">
               <mx:columns>
                  <mx:DataGridColumn headerText="Name"
                     dataField="name"/>
                  <mx:DataGridColumn headerText="Description"
                     dataField="description"/>
               </mx:columns>
            </mx:DataGrid>
         </s:Panel>
      </s:VGroup>
   </s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, this will produce following result: [Try it online]


C:\Users\GB3824\Adobe Flash
C:\Users\GB3824\Adobe Fla...
File Edit View Favorites Tools Help

RPC Service Demonstration

Using RPC Services

Name	Description
Book	History of France
Pen	Parker Pen
Pencil	Stationary

CHAPTER

18

Flex – Flex Unit Integration

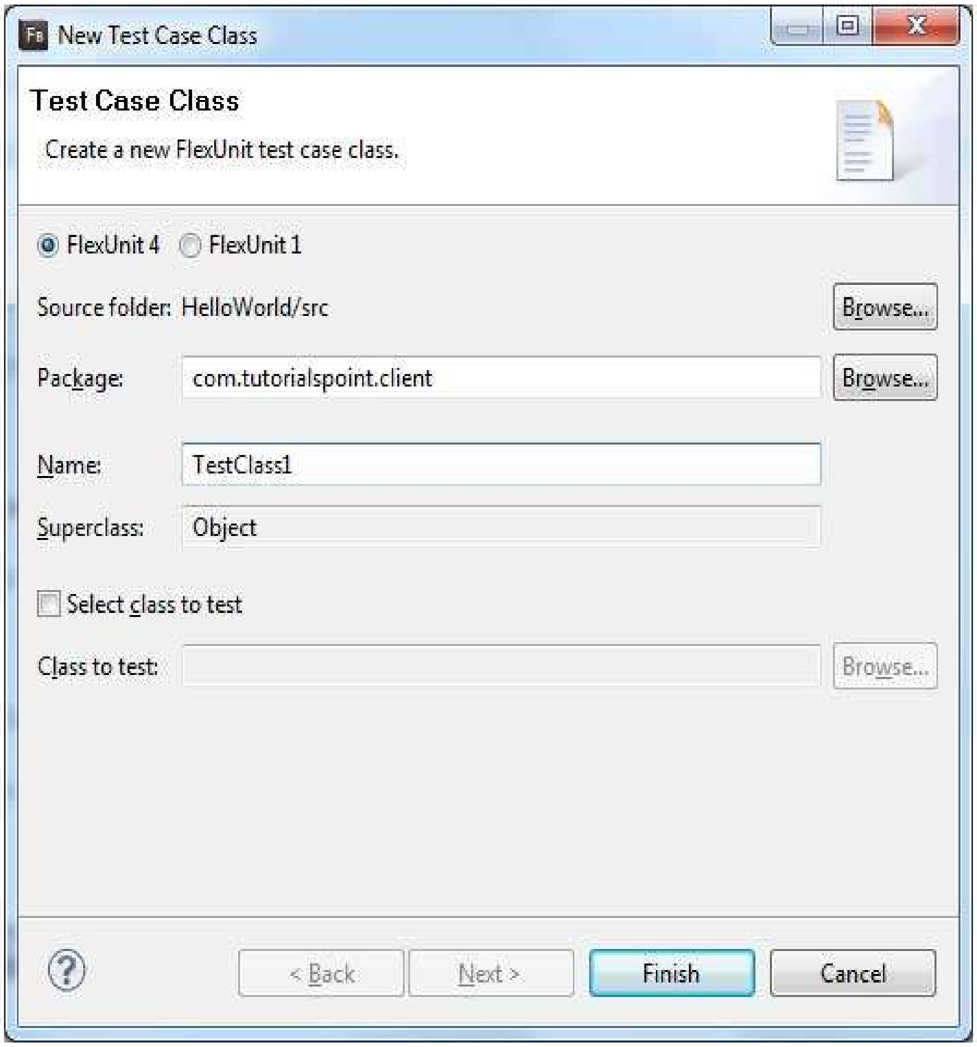
This chapter shows FlexUnit Integration in Flex Development Cycle:.

Flash Builder 4 exececellent inbuilt support for FlexUnit integration in Flex development Cycle.

Create a Test Case Class

You can create a Test Case Class using Flash Builder Create Test Class wizard. Running test cases is a breeze with Flash Builder as you will see in this article.

To create a test case class using Flash Builder, Click on ***File > New > Test Case Class***. Enter the details as shown below.


New Test Case Class

Test Case Class

Create a new FlexUnit test case class.

(● FlexUnit 4) ○ FlexUnit 1

Source folder: HelloWorld/src

Package: com.tutorialspoint.client

Name: TestClass1

Superclass: Object

☐ Select class to test

Class to test:

Flash Builder will create the following TestClass1.as file.

```
package com.tutorialspoint.client
{
   public class TestClass1
   {
      [Before]
      public function setUp():void {}

      [After]
      public function tearDown():void {}

      [BeforeClass]
      public static function setUpBeforeClass():void {}

      [AfterClass]
      public static function tearDownAfterClass():void {}
   }
}
```

FlexUnit Integration Example

Now Let us follow the following steps to test FlexUnit Integration in a Flex application:

Step	Description
1	Create a project with a name *HelloWorld* under a package *com.tutorialspoint.client* as explained in the *Flex - Create Application* chapter.
2	Modify *HelloWorld.mxml* as explained below. Keep rest of the files unchanged.
3	Create *TestClass1.as* test case as described above and Modify *TestClass1.as* as explained below.
4	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified as file **src/com.tutorialspoint/client/TestClass1.as**.

```
package com.tutorialspoint.client
{
   import org.flexunit.asserts.assertEquals;

   public class TestClass1
   {
      private var counter: int = 1;

      [Before]
      public function setUp():void
      {
         //this code will run before every test case execution
      }

      [After]
      public function tearDown():void
      {
         //this code will run after every test case execution
      }

      [BeforeClass]
      public static function setUpBeforeClass():void
      {
         //this code will run once when test cases start execution
      }

      [AfterClass]
      public static function tearDownAfterClass():void
      {
         //this code will run once when test cases ends execution
      }

      [Test]
      public function testCounter():void {
         assertEquals(counter, 1);
      }
   }
}
```

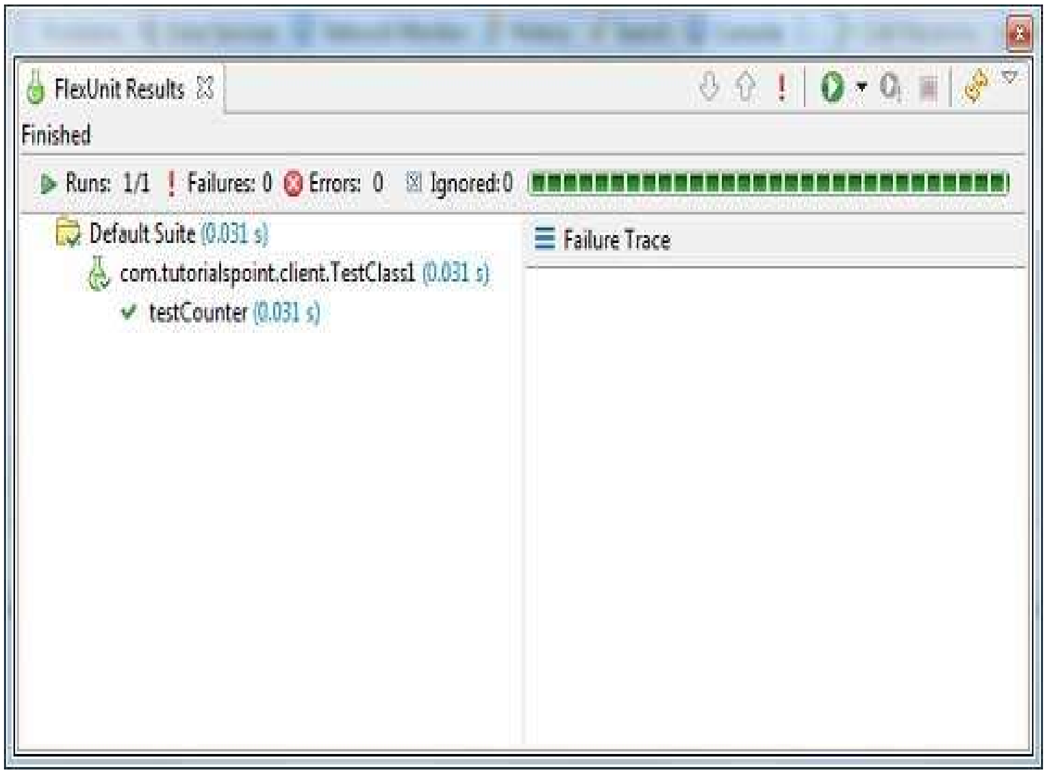
Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   minWidth="500" minHeight="500">
</s:Application>
```

Once you are ready with all the changes done, let us compile in normal mode as we did in Flex - Create Application chapter.

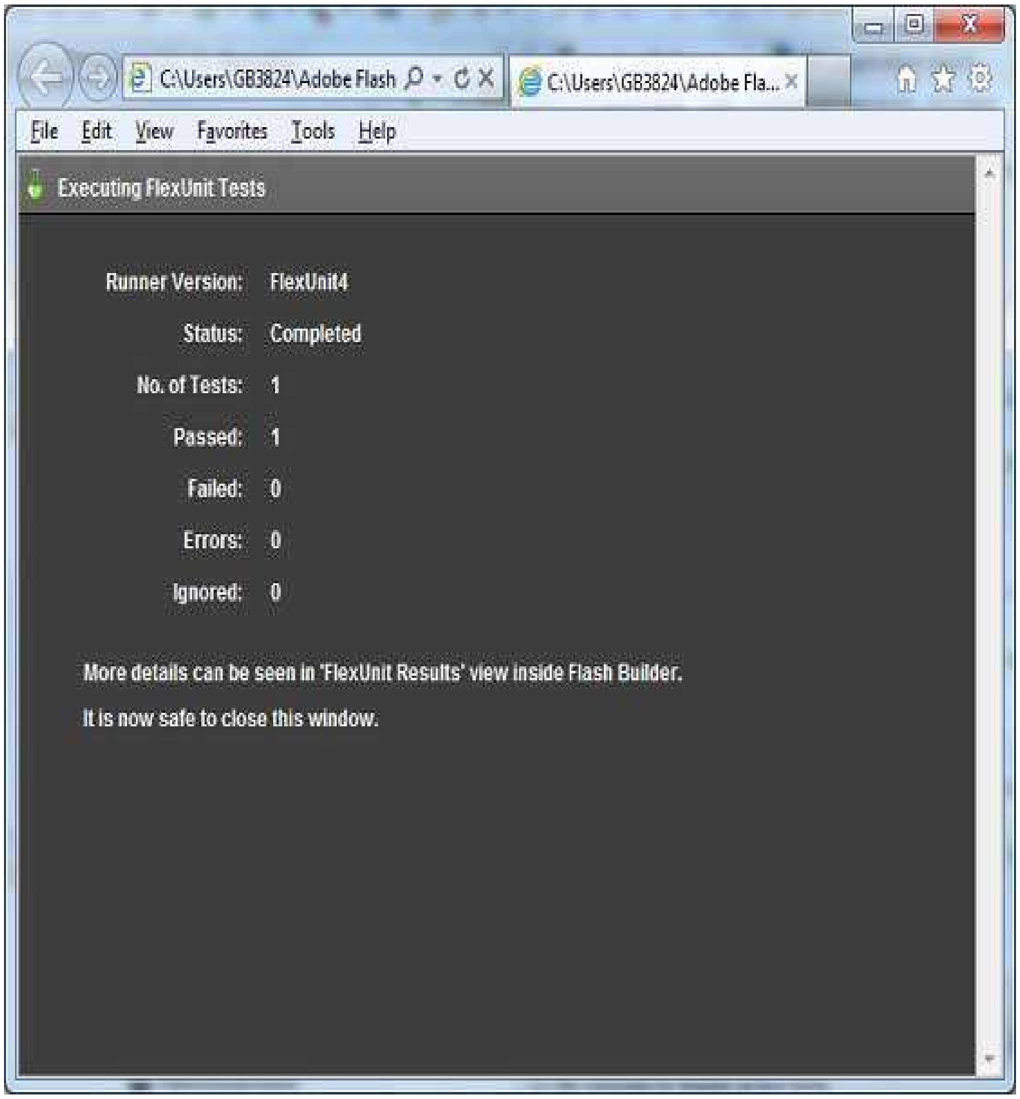
Running Test cases

Now Right Click on TestClass1 in package explorer and select ***Run As > FlexUnit Tests***. You'll see the following output in Flash Builder test window.



```
Default Suite (0.031 s)
  com.tutorialspoint.client.TestClass1 (0.031 s)
    testCounter (0.031 s)
```

Flash Builder also show test result in the browser.


C:\Users\GB3824\Adobe Flash
C:\Users\GB3824\Adobe Fla...
File Edit View Favorites Tools Help

Executing FlexUnit Tests

```
Runner Version:  FlexUnit4
        Status:  Completed
  No. of Tests:  1
        Passed:  1
        Failed:  0
        Errors:  0
       Ignored:  0
```

More details can be seen in 'FlexUnit Results' view inside Flash Builder.

It is now safe to close this window.

CHAPTER

19

Flex – Debug Application

This chapter describes the debugging application of Adobe Flex::.

Flex provides execellent capability of debugging flex code and Flash Builder 4 has an execellent built-in debugger and debugging perspective support.

- During debug mode, Flex Application runs on Flash Player Debugger version built in Flash Builder 4 which supports debugging capabilibity.
- So developers get an easy and inbuilt debugging configuration in Flash Builder

  In this article we'll demonstrate usage of debugging Flex Client code using Flash Builder. We'll do the following tasks

- Set break points in the code and see them in BreakPoint Explorer.
- Step through the code line by line during debugging.
- View the values of variable.
- Inspect the values of all the variables.
- Inspect the value of an expression.
- Display the stack frame for suspended threads.

Debugging Example

Step	Description
1	Create a project with a name *HelloWorld* under a package *com.tutorialspoint.client* as explained in the *Flex - Create Application* chapter.
2	Modify *HelloWorld.mxml* as explained below. Keep rest of the files unchanged.

3	Compile and run the application to make sure business logic is working as per the requirements.

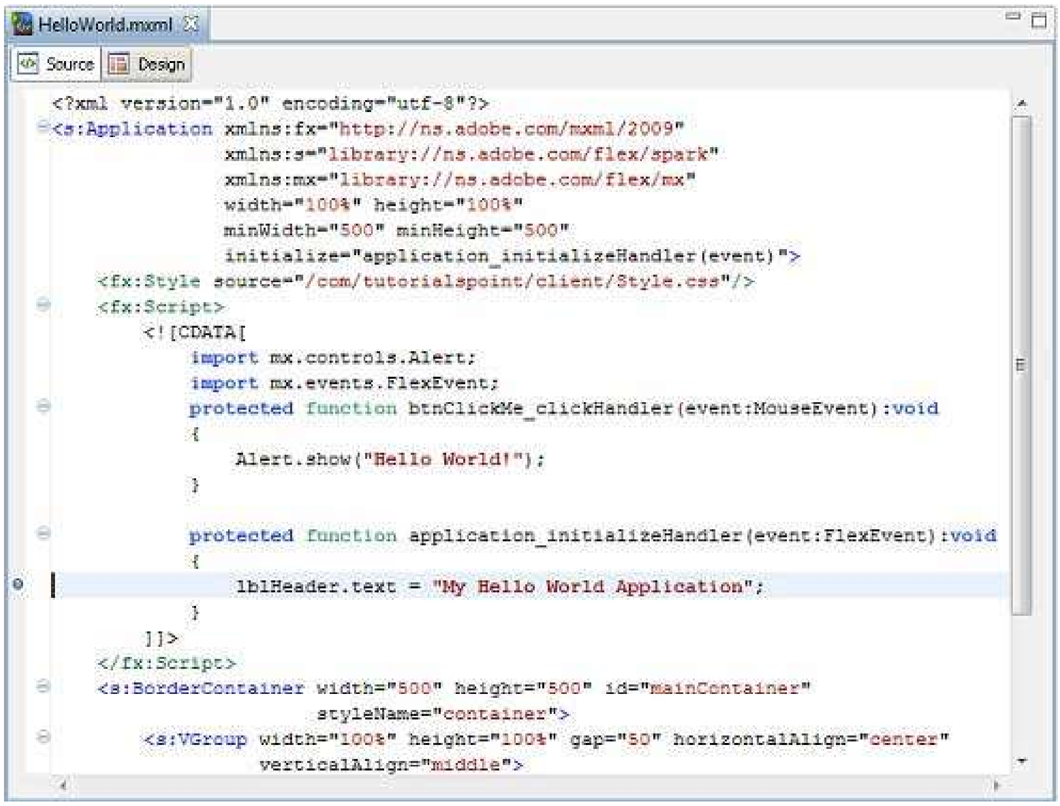
Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   width="100%" height="100%"
   minWidth="500" minHeight="500"
   initialize="application_initializeHandler(event)">
   <fx:Style source="/com/tutorialspoint/client/Style.css"/>
   <fx:Script>
      <![CDATA[
       import mx.controls.Alert;
       import mx.events.FlexEvent;
       protected function btnClickMe_clickHandler(event:MouseEvent):void
       {
          Alert.show("Hello World!");
       }

       protected function application_initializeHandler(event:FlexEvent):void
       {
          lblHeader.text = "My Hello World Application";
       }
      ]]>
   </fx:Script>
   <s:BorderContainer width="500" height="500" id="mainContainer"
      styleName="container">
      <s:VGroup width="100%" height="100%" gap="50" horizontalAlign="center"
         verticalAlign="middle">
         <s:Label id="lblHeader" fontSize="40" color="0x777777"
            styleName="heading"/>
         <s:Button label="Click Me!" id="btnClickMe"
            click="btnClickMe_clickHandler(event)" styleName="button" />
      </s:VGroup>
   </s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile in normal mode as we did in Flex - Create Application chapter.

Step 1 - Place BreakPoints

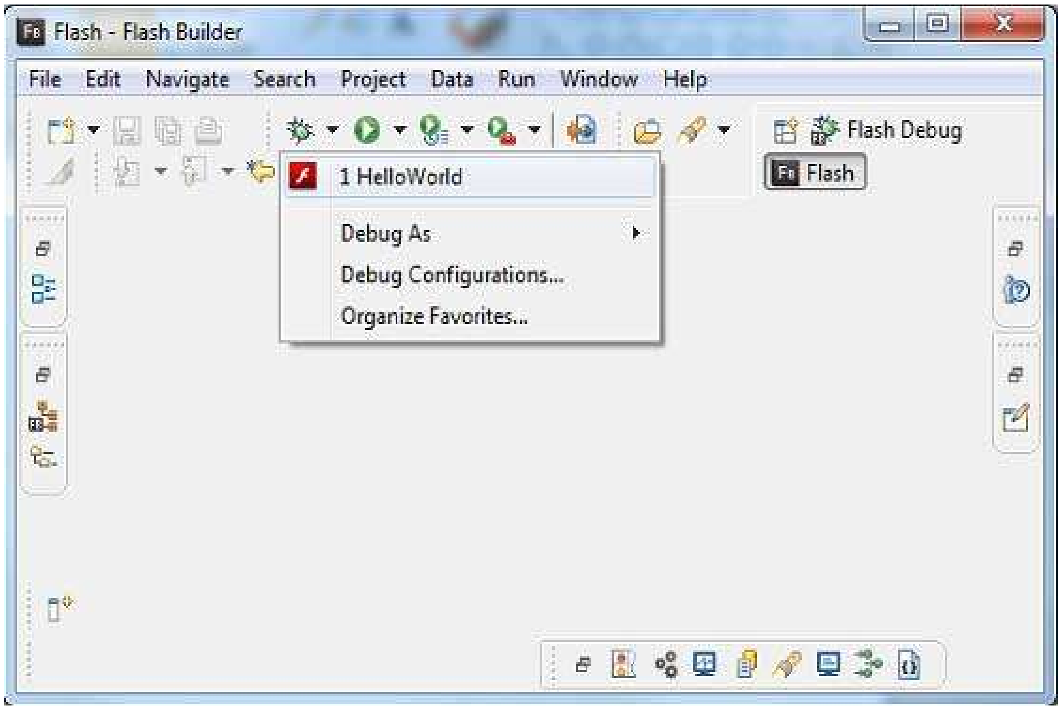
Place a breakpoint on the first line of application_initializeHandler of HelloWorld.mxml



```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx"
               width="100%" height="100%"
               minWidth="500" minHeight="500"
               initialize="application_initializeHandler(event)">
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.events.FlexEvent;
            protected function btnClickMe_clickHandler(event:MouseEvent):void
            {
                Alert.show("Hello World!");
            }

            protected function application_initializeHandler(event:FlexEvent):void
            {
                lblHeader.text = "My Hello World Application";
            }
        ]]>
    </fx:Script>
    <s:BorderContainer width="500" height="500" id="mainContainer"
                       styleName="container">
        <s:VGroup width="100%" height="100%" gap="50" horizontalAlign="center"
                  verticalAlign="middle">
```

Step 2 - Debug Application

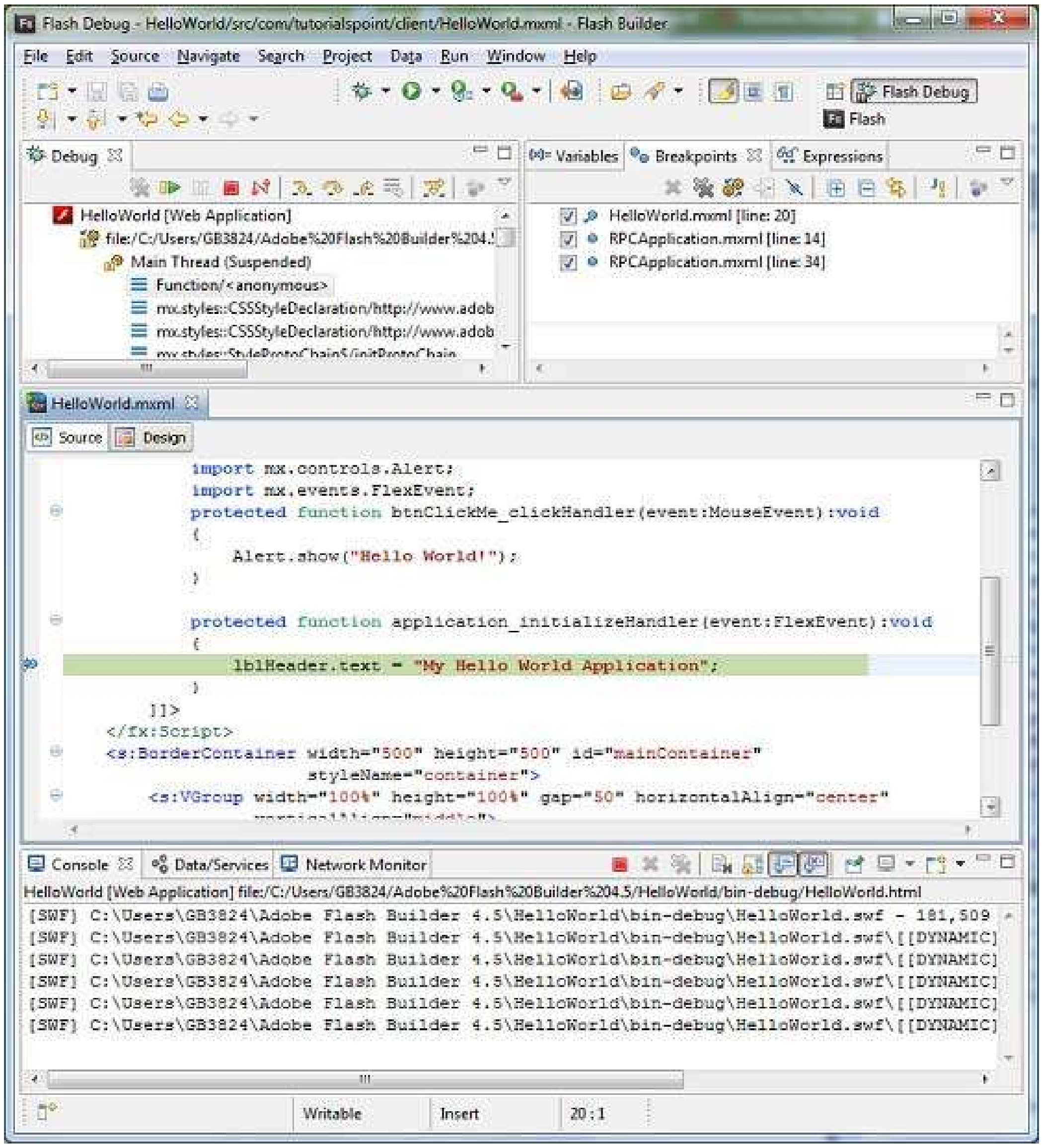
Now click on Debug application menu and select **HelloWorld** application to debug the application.



If everything is fine, application will launch in the browser and you will see following debug logs in Flash Builder console

```
[SWF] \HelloWorld\bin-debug\HelloWorld.swf
- 181,509 bytes after decompression
[SWF] \HelloWorld\bin-debug\HelloWorld.swf\[[DYNAMIC]]\1
- 763,122 bytes after decompression
[SWF] \HelloWorld\bin-debug\HelloWorld.swf\[[DYNAMIC]]\2
- 1,221,837 bytes after decompression
[SWF] \HelloWorld\bin-debug\HelloWorld.swf\[[DYNAMIC]]\3
- 1,136,788 bytes after decompression
[SWF] \HelloWorld\bin-debug\HelloWorld.swf\[[DYNAMIC]]\4
- 2,019,570 bytes after decompression
[SWF] \HelloWorld\bin-debug\HelloWorld.swf\[[DYNAMIC]]\5
- 318,334 bytes after decompression
```

As soon as Application launches,you will see the focus on Flash Builder breakpoint as we've placed the breakpoint on first line of application_initializeHandler method.


Flash Debug - HelloWorld/src/com/tutorialspoint/client/HelloWorld.mxml - Flash Builder
File Edit Source Navigate Search Project Data Run Window Help
Flash Debug
Flash
Debug
Variables Breakpoints Expressions
HelloWorld.mxml Source Design
Console Data/Services Network Monitor
Writable Insert 20 : 1

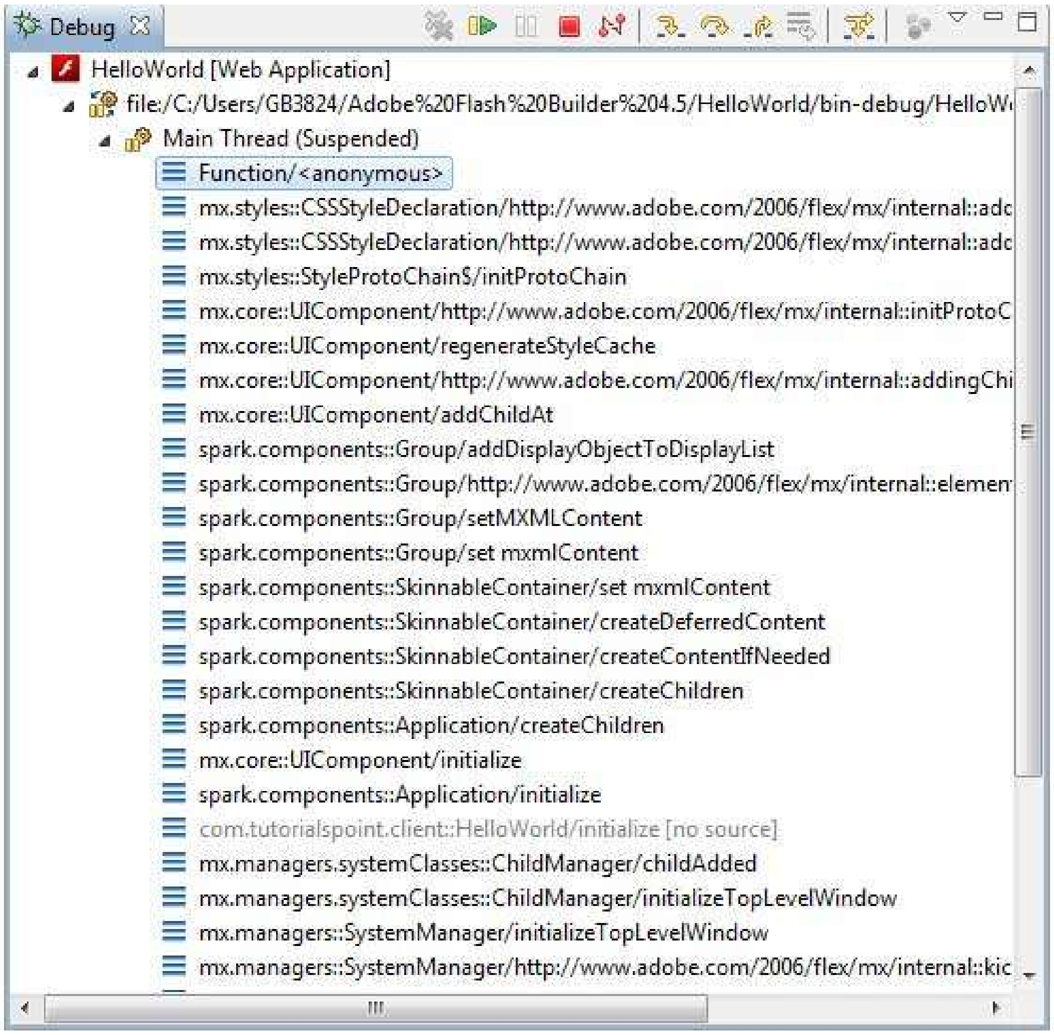
```
HelloWorld [Web Application]
  file:/C:/Users/GB3824/Adobe%20Flash%20Builder%204.5
    Main Thread (Suspended)
      Function/<anonymous>
      mx.styles::CSSStyleDeclaration/http://www.adob
      mx.styles::CSSStyleDeclaration/http://www.adob
```

```
HelloWorld.mxml [line: 20]
RPCApplication.mxml [line: 14]
RPCApplication.mxml [line: 34]
```

```
            import mx.controls.Alert;
            import mx.events.FlexEvent;
            protected function btnClickMe_clickHandler(event:MouseEvent):void
            {
                Alert.show("Hello World!");
            }

            protected function application_initializeHandler(event:FlexEvent):void
            {
                lblHeader.text = "My Hello World Application";
            }
        ]]>
    </fx:Script>
    <s:BorderContainer width="500" height="500" id="mainContainer"
                       styleName="container">
        <s:VGroup width="100%" height="100%" gap="50" horizontalAlign="center"
```

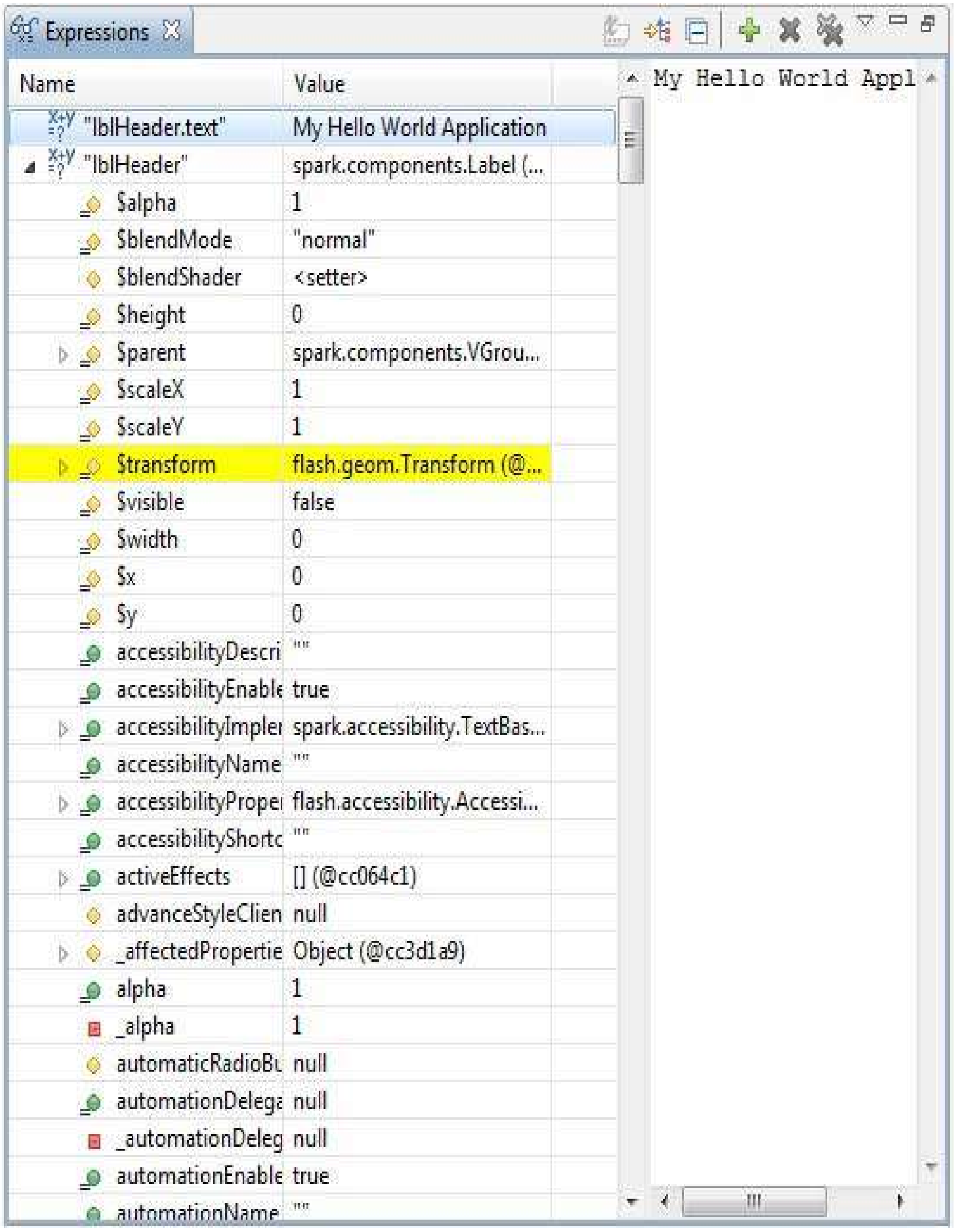
```
HelloWorld [Web Application] file:/C:/Users/GB3824/Adobe%20Flash%20Builder%204.5/HelloWorld/bin-debug/HelloWorld.html
[SWF] C:\Users\GB3824\Adobe Flash Builder 4.5\HelloWorld\bin-debug\HelloWorld.swf - 181,509
[SWF] C:\Users\GB3824\Adobe Flash Builder 4.5\HelloWorld\bin-debug\HelloWorld.swf\[[DYNAMIC]
[SWF] C:\Users\GB3824\Adobe Flash Builder 4.5\HelloWorld\bin-debug\HelloWorld.swf\[[DYNAMIC]
[SWF] C:\Users\GB3824\Adobe Flash Builder 4.5\HelloWorld\bin-debug\HelloWorld.swf\[[DYNAMIC]
[SWF] C:\Users\GB3824\Adobe Flash Builder 4.5\HelloWorld\bin-debug\HelloWorld.swf\[[DYNAMIC]
[SWF] C:\Users\GB3824\Adobe Flash Builder 4.5\HelloWorld\bin-debug\HelloWorld.swf\[[DYNAMIC]
```

You can see the stacktrace for suspended threads.



```
HelloWorld [Web Application]
  file:/C:/Users/GB3824/Adobe%20Flash%20Builder%204.5/HelloWorld/bin-debug/HelloW
    Main Thread (Suspended)
      Function/<anonymous>
      mx.styles::CSSStyleDeclaration/http://www.adobe.com/2006/flex/mx/internal::add
      mx.styles::CSSStyleDeclaration/http://www.adobe.com/2006/flex/mx/internal::add
      mx.styles::StyleProtoChain$/initProtoChain
      mx.core::UIComponent/http://www.adobe.com/2006/flex/mx/internal::initProtoC
      mx.core::UIComponent/regenerateStyleCache
      mx.core::UIComponent/http://www.adobe.com/2006/flex/mx/internal::addingChi
      mx.core::UIComponent/addChildAt
      spark.components::Group/addDisplayObjectToDisplayList
      spark.components::Group/http://www.adobe.com/2006/flex/mx/internal::elemen
      spark.components::Group/setMXMLContent
      spark.components::Group/set mxmlContent
      spark.components::SkinnableContainer/set mxmlContent
      spark.components::SkinnableContainer/createDeferredContent
      spark.components::SkinnableContainer/createContentIfNeeded
      spark.components::SkinnableContainer/createChildren
      spark.components::Application/createChildren
      mx.core::UIComponent/initialize
      spark.components::Application/initialize
      com.tutorialspoint.client::HelloWorld/initialize [no source]
      mx.managers.systemClasses::ChildManager/childAdded
      mx.managers.systemClasses::ChildManager/initializeTopLevelWindow
      mx.managers::SystemManager/initializeTopLevelWindow
      mx.managers::SystemManager/http://www.adobe.com/2006/flex/mx/internal::kic
```

You can see the values for expressions.

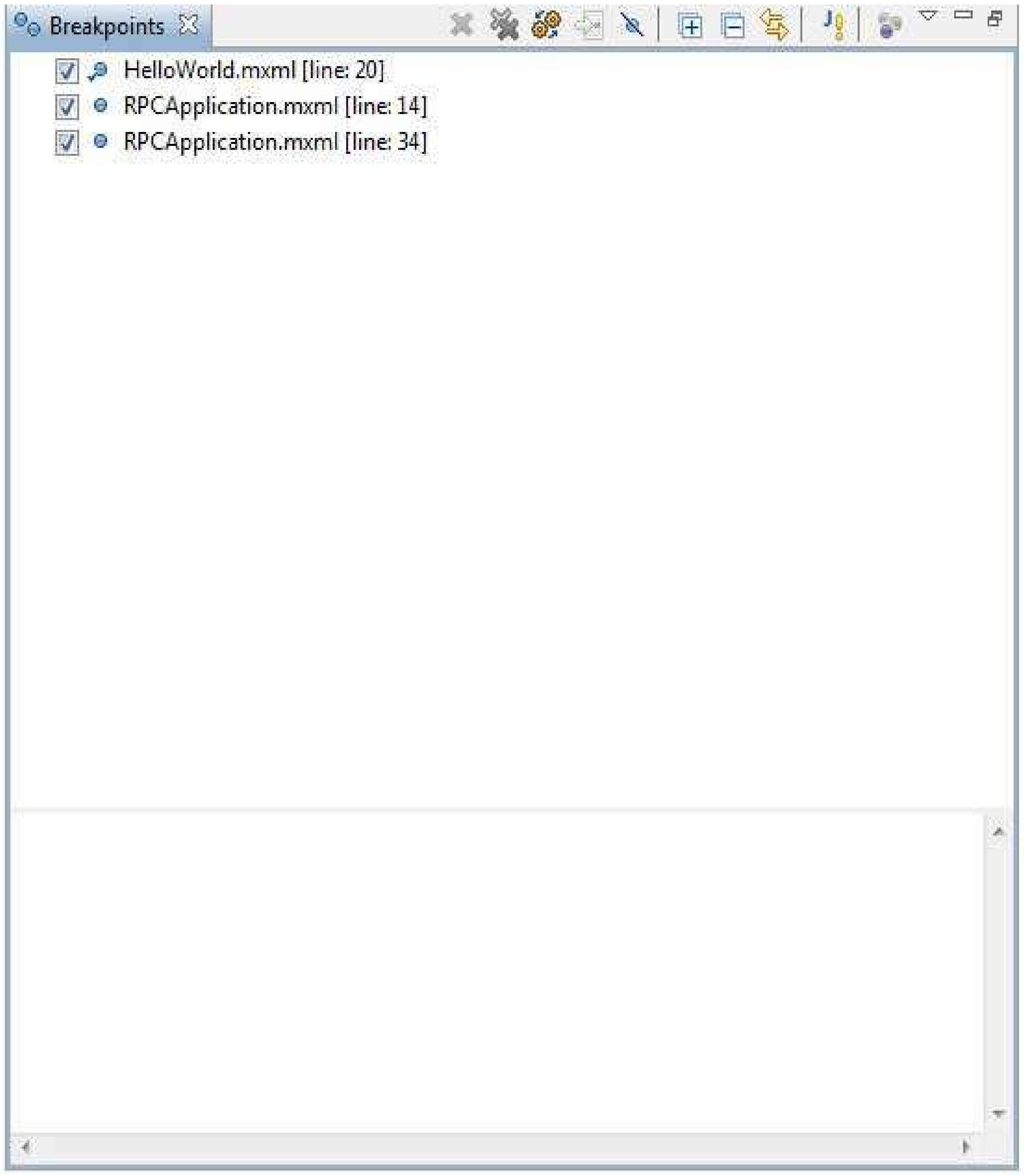


Expressions

Name	Value
"lblHeader.text"	My Hello World Application
"lblHeader"	spark.components.Label (...
$alpha	1
$blendMode	"normal"
$blendShader	<setter>
$height	0
$parent	spark.components.VGrou...
$scaleX	1
$scaleY	1
$transform	flash.geom.Transform (@...
$visible	false
$width	0
$x	0
$y	0
accessibilityDescri	""
accessibilityEnable	true
accessibilityImpler	spark.accessibility.TextBas...
accessibilityName	""
accessibilityProper	flash.accessibility.Accessi...
accessibilityShortc	""
activeEffects	[] (@cc064c1)
advanceStyleClien	null
_affectedPropertie	Object (@cc3d1a9)
alpha	1
_alpha	1
automaticRadioBu	null
automationDelega	null
_automationDeleg	null
automationEnable	true
automationName	""

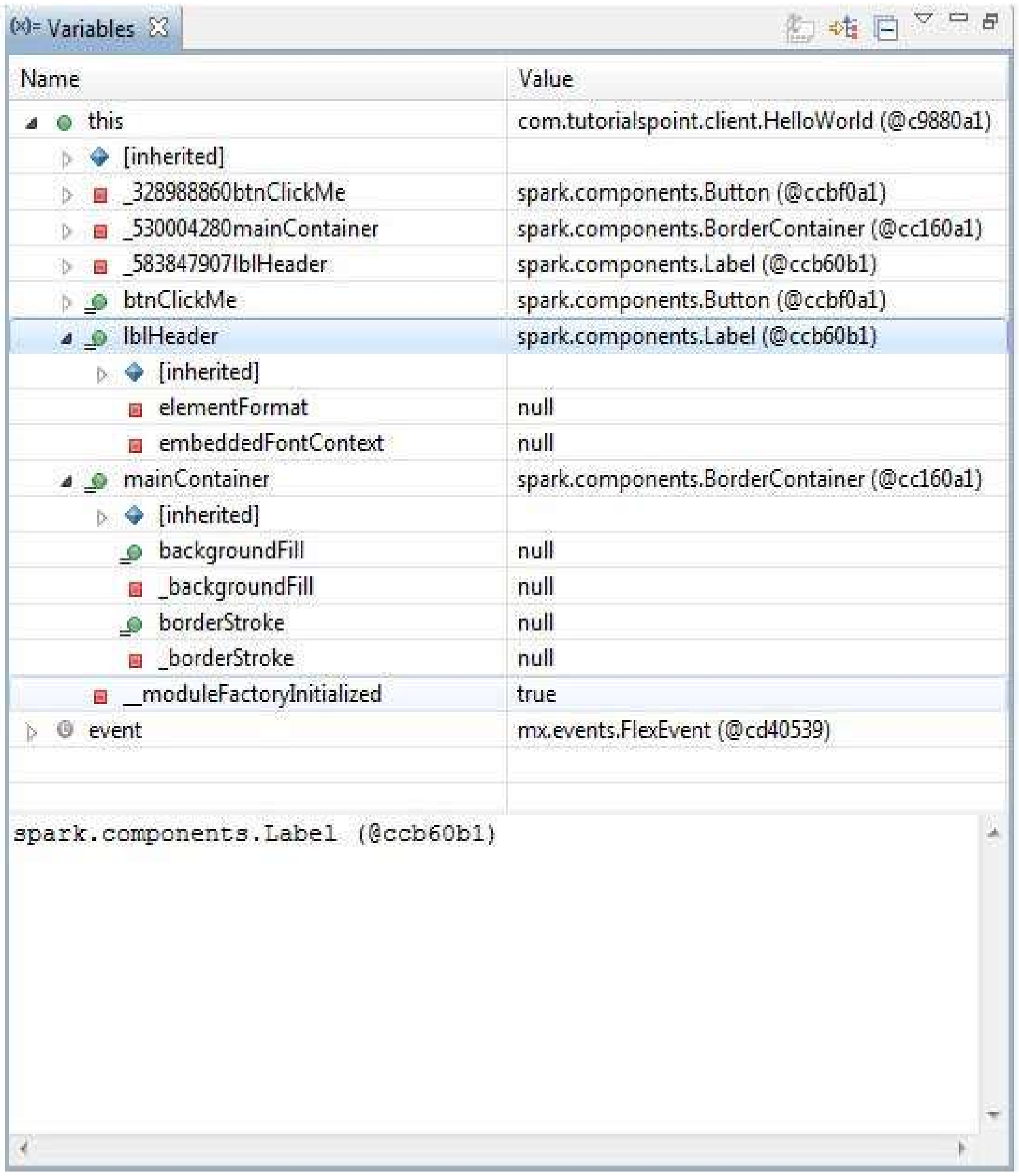
```
My Hello World Appl
```

You can see the list of breakpoints placed.



HelloWorld.mxml [line: 20]
RPCApplication.mxml [line: 14]
RPCApplication.mxml [line: 34]

Now keep pressing F6 until you reach the last line of application_initializeHandler() method. As reference for function keys, F6 inspects code line by line, F5 steps inside further and F8 will resume the application. Now you can see the list of values of all variables of application_initializeHandler() method.



Name	Value
this	com.tutorialspoint.client.HelloWorld (@c9880a1)
[inherited]	
_328988860btnClickMe	spark.components.Button (@ccbf0a1)
_530004280mainContainer	spark.components.BorderContainer (@cc160a1)
_583847907lblHeader	spark.components.Label (@ccb60b1)
btnClickMe	spark.components.Button (@ccbf0a1)
lblHeader	spark.components.Label (@ccb60b1)
[inherited]	
elementFormat	null
embeddedFontContext	null
mainContainer	spark.components.BorderContainer (@cc160a1)
[inherited]	
backgroundFill	null
_backgroundFill	null
borderStroke	null
_borderStroke	null
__moduleFactoryInitialized	true
event	mx.events.FlexEvent (@cd40539)

```
spark.components.Label (@ccb60b1)
```

Now you can see the flex code can be debugged in the same way as a Java Application can be debugged. Place breakpoints to any line and play with debugging capabilities of flex.

CHAPTER

20

Flex - Internationalization

This chapter describes internalization of Flex aapplication:

Flex provides two ways to internationalize a Flex application, We'll demonstrate use of Compile time Internationalization being most commonly used among projects.

Technique	Description
Compile Time Internationalization	This technique is most prevalent and requires very little overhead at runtime; is a very efficient technique for translating both constant and parameterized strings;simplest to implement. Compile Time internationalization uses standard properties files to store translated strings and parameterized messages, and these properties files are compiled directly in the application.
Run Time Internationalization	This technique is very flexible but slower than static string internationalization. You need to compile the localization properties files separately, leave them external to application, and load them at run time.

Workflow of internationalizing a Flex Application

Step 1: Create folder structure

Create a locale folder under src folder of Flex project.This will be the parent directory for all of the properties files for the locales that the application will support. Inside the locale folder, create subfolders, one for each of the application's locales to be supported. The convention for naming a locale is

```
{language}_{country code}
```

For example, en_US represents English of the United States. The locale de_DE represents German. The sample application will support two common languages: English, and German

Step 2: Create properties files

Create properties file containing the messages to be used in the application. We've created a**HelloWorldMessages.properties** file under ***src > locale > en_US*** folder in our example.

```
enterName=Enter your name
clickMe=Click Me
applicationTitle=Application Internationalization Demonstration
greeting=Hello {0}
```

Create properties files containing translated values specific to locale. We've created a**HelloWorldMessages.properties** file under ***src > locale > de_DE*** folder in our example. This file contains translations in german language. _de specifies the german locale and we're going to support german language in our application.

If you are creating properties file using Flash Builder then change the encoding of the file to UTF-8.Select the file and then right-click in it to open its properties window.Select Text file encoding as**Other UTF-8**. Apply and Save the change.

```
enterName=Geben Sie Ihren Namen
clickMe=Klick mich
applicationTitle=Anwendung Internationalisierung Demonstration
greeting=Hallo {0}
```

Step 3: Specify Compiler options

- Right-click your project and select Properties.
- Select Flex Compiler, and add the following to the Additional Compiler Arguments settings:

```
-locale en_US de_DE
```

- Right-click your project and select Properties.
- Select Flex Build Path, and add the following to the Source Path settings:

```
src\locale\{locale}
```

Internalization Example

Now Let us follow the following steps to test Internalization technique in a Flex application:

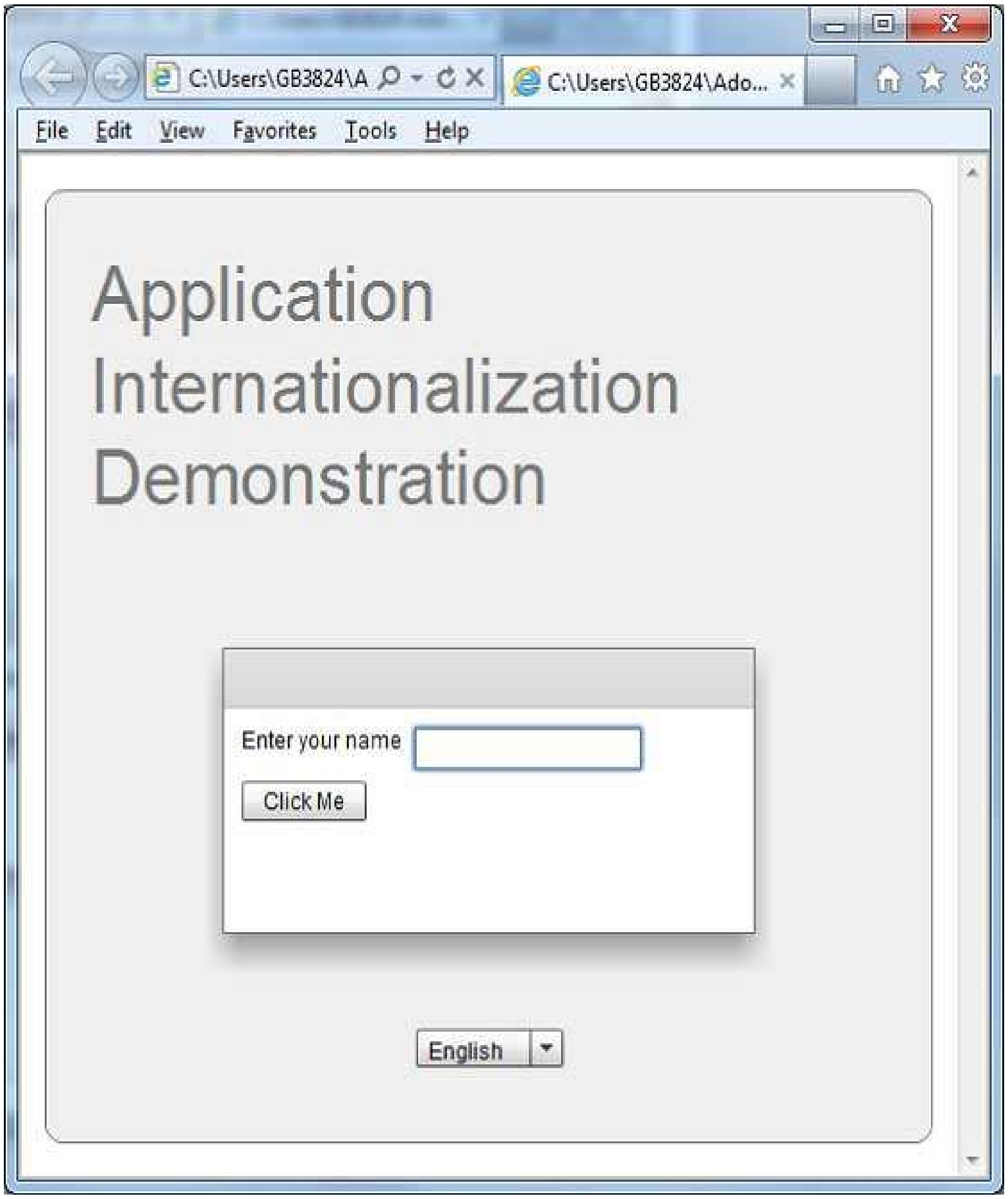
Step	Description
1	Create a project with a name *HelloWorld* under a package *com.tutorialspoint.client* as explained in the *Flex - Create Application* chapter.

2	Modify *HelloWorld.mxml* as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

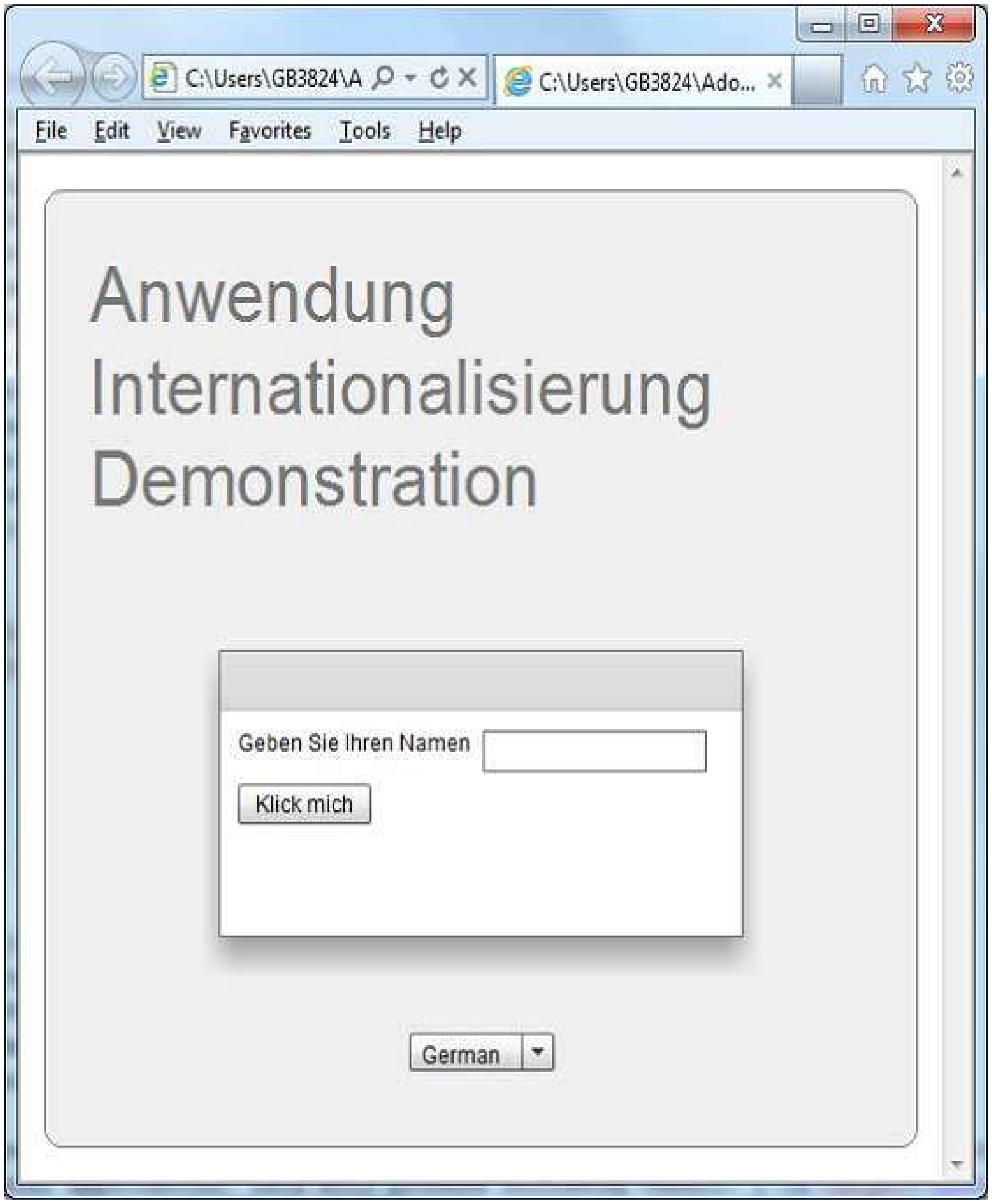
```
<?xml version="1.0" encoding="utf-8"?>
   <s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   minWidth="500" minHeight="500">
   <fx:Metadata>
      [ResourceBundle("HelloWorldMessages")]
   </fx:Metadata>
   <fx:Style source="/com/tutorialspoint/client/Style.css"/>
   <fx:Script>
      <![CDATA[
         import mx.controls.Alert;
         [Bindable]
         private var locales:Array = [{label:"English", locale:"en_US"},
            {label:"German", locale:"de_DE"}];

         private function comboChangeHandler():void
         {
           resourceManager.localeChain = [localeComboBox.selectedItem.locale];
         }

         protected function clickMe_clickHandler(event:MouseEvent):void
         {
            var name:String = txtName.text;
            var inputArray:Array = new Array();
            inputArray.push(name);
            Alert.show(resourceManager.getString('HelloWorldMessages'
            ,'greeting',inputArray));
         }
      ]]>
   </fx:Script>
   <s:BorderContainer width="500" height="500" id="mainContainer"
      styleName="container">
      <s:VGroup width="100%" height="100%" gap="50"
                          horizontalAlign="center" verticalAlign="middle">
         <s:Label id="lblHeader" fontSize="40"
            color="0x777777"
            text ="{resourceManager.getString('HelloWorldMessages'
            ,'applicationTitle')}"
            styleName="heading" width="90%" height="150"/>
         <s:Panel width="300" height="150">
            <s:layout>
               <s:VerticalLayout paddingTop="10" paddingLeft="10" />
            </s:layout>
            <s:HGroup >
               <s:Label
               text="{resourceManager.getString('HelloWorldMessages'
               ,'enterName')}"
               paddingTop="2"/>
               <s:TextInput id="txtName"/>
            </s:HGroup>
            <s:Button
            label="{resourceManager.getString('HelloWorldMessages','clickMe')}"
            click="clickMe_clickHandler(event)" right="10" />
         </s:Panel>
         <mx:ComboBox id="localeComboBox"
         dataProvider="{locales}" change="comboChangeHandler()"/>
      </s:VGroup>
   </s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, this will produce following result: [Try it online]



Application Internationalization Demonstration

Change the language using language drop down and see the result.



Anwendung Internationalisierung Demonstration

Geben Sie Ihren Namen

Klick mich

German

CHAPTER

21

Flex – Printing Support

Flex provides a special class **FlexPrintJob** to print flex objects.

- FlexPrintJob can be used to print one or more Flex objects, such as a Form or VBox container.
- FlexPrintJob prints the object and all objects that it contains.
- The objects can be all or part of the displayed interface.
- The objects can be components that format data specifically for printing.
- The FlexPrintJob class lets you scale the output to fit the page.
- The FlexPrintJob class automatically uses multiple pages to print an object that does not fit on a single page.
- The FlexPrintJob class causes the operating system to display a Print dialog box. You cannot print without some user action.

Prepare and send a print job

You print output by preparing and sending a print job. Let's create an instance of the FlexPrintJob class

```
var printJob:FlexPrintJob = new FlexPrintJob();
```

Start the print job

```
printJob.start();
```

Flex will cause the operating system to display a Print dialog box. Add one or more objects to the print job and specify how to scale them

```
printJob.addObject(myObject, FlexPrintJobScaleType.MATCH_WIDTH);
```

Each object starts on a new page. Send the print job to the printer

```
printJob.send();
```

Printing Example

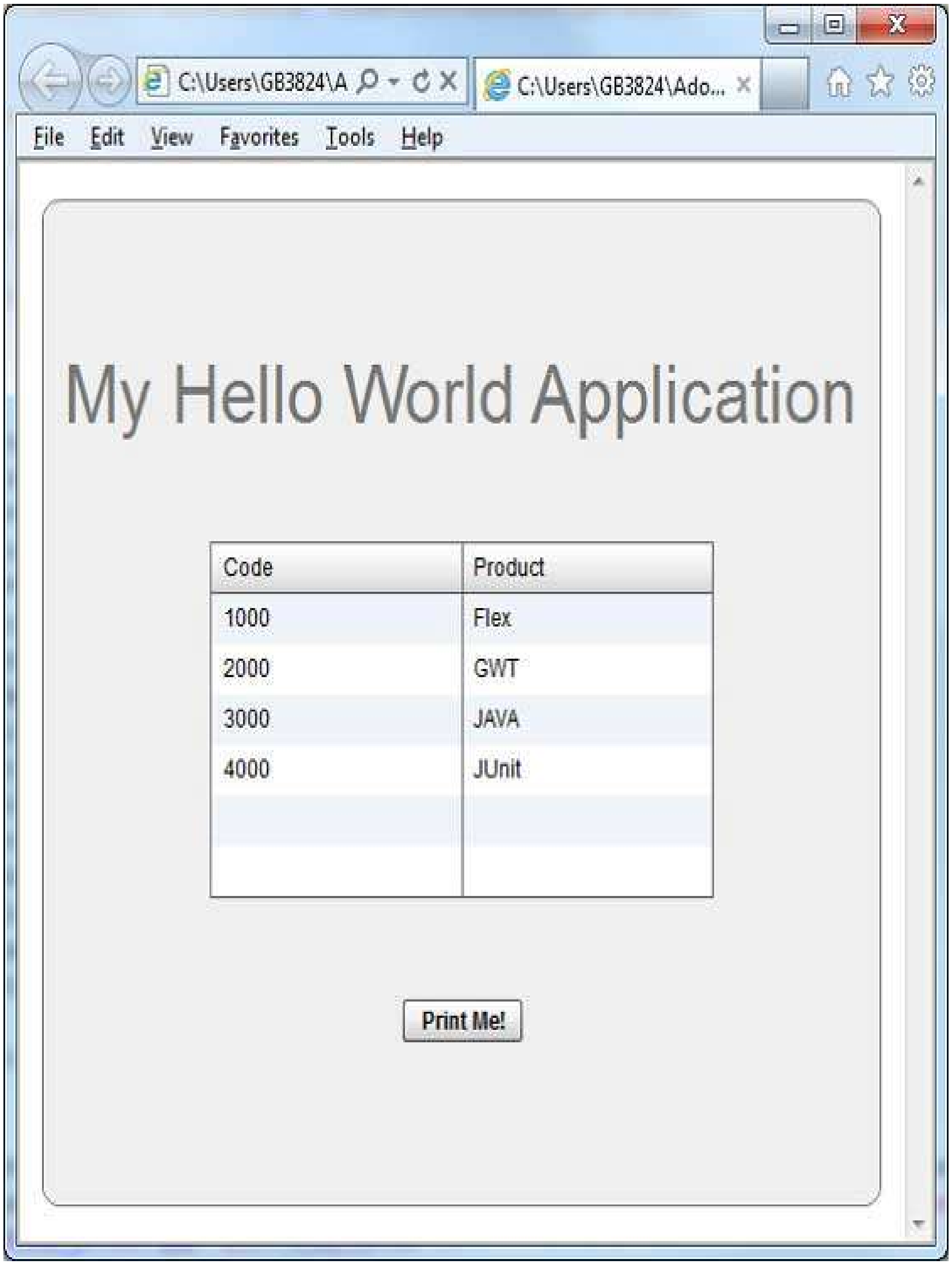
Step	Description
1	Create a project with a name *HelloWorld* under a package *com.tutorialspoint.client* as explained in the *Flex - Create Application* chapter.
2	Modify *HelloWorld.mxml* as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   width="100%" height="100%"
   minWidth="500" minHeight="500"
   initialize="application_initializeHandler(event)">
   <fx:Style source="/com/tutorialspoint/client/Style.css"/>
   <fx:Script>
     <![CDATA[
        import mx.controls.Alert;
        import mx.events.FlexEvent;
        import mx.printing.FlexPrintJob;
        import mx.printing.FlexPrintJobScaleType;
        protected function btnClickMe_clickHandler(event:MouseEvent):void
        {
            // Create an instance of the FlexPrintJob class.
            var printJob:FlexPrintJob = new FlexPrintJob();

            // Start the print job.
            if (printJob.start() != true) return;

            // Add the object to print. Do not scale it.
            printJob.addObject(myDataGrid, FlexPrintJobScaleType.NONE);

            // Send the job to the printer.
            printJob.send();
        }

        protected function application_initializeHandler(event:FlexEvent):void
        {
            lblHeader.text = "My Hello World Application";
        }
     ]]>
   </fx:Script>
   <s:BorderContainer width="500" height="500" id="mainContainer"
      styleName="container">
      <s:VGroup width="100%" height="100%" gap="50"
```

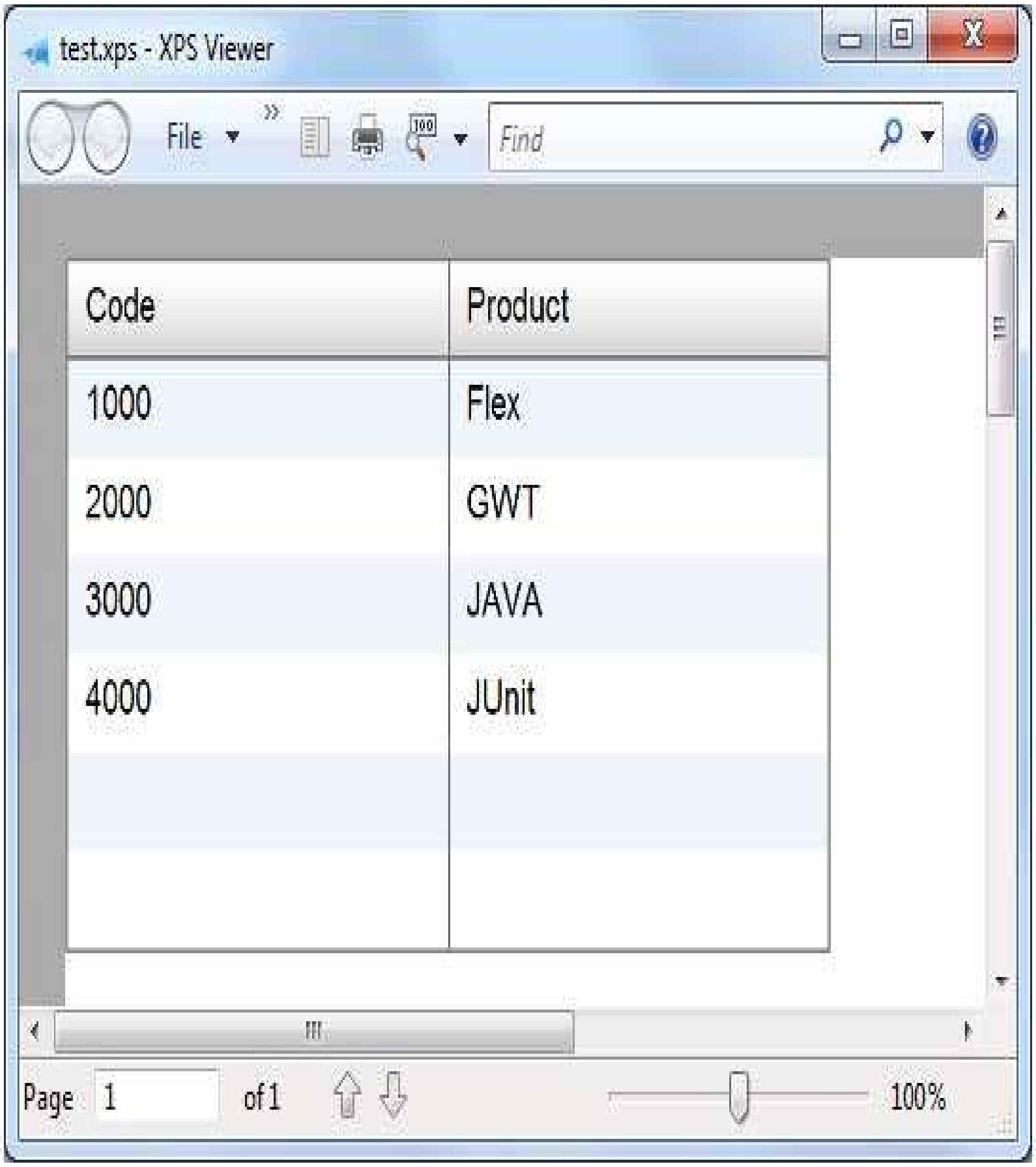
```
         horizontalAlign="center"
         verticalAlign="middle">
         <s:Label id="lblHeader" fontSize="40" color="0x777777"
            styleName="heading"/>
         <mx:DataGrid id="myDataGrid" width="300">
            <mx:dataProvider>
               <fx:Object Product="Flex" Code="1000"/>
               <fx:Object Product="GWT" Code="2000"/>
               <fx:Object Product="JAVA" Code="3000"/>
               <fx:Object Product="JUnit" Code="4000"/>
            </mx:dataProvider>
         </mx:DataGrid>
         <s:Button label="Print Me!" id="btnClickMe"
            click="btnClickMe_clickHandler(event)"
            styleName="button" />
      </s:VGroup>
   </s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, this will produce following result: [Try it online]


C:\Users\GB3824\A
C:\Users\GB3824\Ado...
File Edit View Favorites Tools Help
My Hello World Application
Print Me!

Code	Product
1000	Flex
2000	GWT
3000	JAVA
4000	JUnit

Click on print me button and you can see the printout of the data grid shown below.



Code	Product
1000	Flex
2000	GWT
3000	JAVA
4000	JUnit